VOLUME 06 | ISSUE 07A

digit FastTrack
YOUR HANDY GUIDE TO EVERYDAY TECHNOLOGY

To Developing Rich Internet Apps

Use **Adobe Flex** and **Flash Builder** to build for multiple mobile platforms, including Android™, BlackBerry® Tablet OS and iOS mobile platforms using a common code base

RIAs a nd **mobileap ps**

Cross-platform **compatible apps**

MXML

ActionScript 3

Applications for **mobile devices**

Flash Builder

Desktop AIR applications

Your first **Flex app**

A 9.9 Media Publication

Check out
Flash Builder Developer Center
and Adobe Developer Connection!

Adobe has always been known as a master software building company which keeps in mind all the needs of the web developers. Now we bring to you Flash Builder Developer Center and Adobe Developer Connection, two unique resource centers where you can know everything about our integrated set of technologies.

To create & deliver the most compelling & expressive applications, rich media content and video to the world, just log on to the links given below and learn all you want to with Flash Builder Developer Center and Adobe Developer Connection.

Start creating the world of your choice with Flash Builder Developer Center and Adobe Developer Connection NOW!

www.adobe.com/devnet/flash-builder.html
www.adobe.com/devnet.html

FAST TRACK to

DEVELOPING RICH INTERNET APPLICATIONS

Use Adobe Flex and Flash Builder to build for multiple mobile platforms, including Android™, BlackBerry® Tablet OS, and iOS mobile platforms using a common code base

powered by

CHAPTERS

RIA | JULY 2011

CREDITS

The People Behind This Book

EDITORIAL

Editor
Robert Sovereign-Smith

Head-Copy Desk
Nash David

Writer
Hanu Prateek Kunduru

DESIGN

Sr. Creative Director
Jayan K Narayanan

Art Director
Binesh Sreedharan

Associate Art Director
Anil VK

Sr. Visualisers
PC Anoop

Senior Designers
Baiju N V, Chander Dange, Vinod Shinde

TO READ THE ABOVE CHAPTER, SCAN THE **QR CODE**, OR GO TO BIT.LY/FT-CH8 OR OPEN THIS MONTH'S DVD

July 2011
Free with Digit. Not to be sold separately. If you have paid separately for this book, please email the editor at editor@thinkdigit.com along with details of location of purchase, for appropriate action.

COVER DESIGN: SHIGIL N.

INTRODUCTION

Development of applications has never gotten as diverse as it is today. Looking back a few years, when people had to build software applications, it would primarily be used on computers and the frameworks were few. The operating systems themselves were restricted to Windows, Mac or Linux. Then with the rise of the internet came the Rich Internet Applications or RIAs which would run on the web or browsers just like desktop applications in many ways. But now with the rapid advancements in mobile technologies, the advent of the modern age smart phones, and their usage & availability picking up like wildfire, the demand for mobile applications is immense. Even statistically with a global figure of over 5 billion mobile users, it is clear that the smartphone market has massive potential for growth. There are a multitude of Operating systems like iOS, Android, Blackberry OS, Windows Phone 7 etc. and different versions of the same. The frameworks and platforms used for developing applications for these devices are also aplenty.

There is a big necessity for finding a single framework to build applications compatible across all devices and operating systems. Adobe has taken a very big step in this direction. With the release of Adobe Flash Builder 4.5 and Flex 4.5 with the Adobe Creative Suite 5.5 release, it is now offering unprecedented new capabilities for building mobile applications and content. These new additions apart from their older application for building software for desktops, now offer a new mobile development paradigm as well, letting you make use of one single tool and development framework for building applications that work across different platforms. What this means is that with one primary code base you can build applications for all major Android, Blackberry Tablet OS and iOS devices.

Throughout the rest of this Fast Track we will be going through the basic languages and tools you will need to build desktop and mobile applications using the Flash platform and the Flex framework. We will be looking at MXML and ActionScript 3 in particular and also Flash and Flash Builder to build RIAs for the web and applications for the desktop. We will learn a little bit about the Adobe AIR runtime and we will also be looking in depth at how to code in these languages using these tools. The final part of this fast track will concentrate on using the same code base to package applications for different mobile devices and platforms.

Mobile Application Development is a very interesting, fun and lucrative field. And the ability to build cross-platform applications can be very advantageous. Therefore the power provided by the Adobe Flex 4.5 framework is amazing. So, all the best with your applications and happy coding.

SCAN THE **QR CODE**
FOR AN INTERESTING
ARTICLE BY ADOBE, OR
GO TO ADOBE.LY/APP-DEV

RIAs AND MOBILE APPS

Latest trends indicate a leaning towards mobile computing. RIAs offer an effective way to cater to the tablet-savvy consumer...

Rich Internet Applications

RIAs or Rich Internet Applications are web applications that work like full-fledged desktop applications in many ways. They are delivered to the user through a site specific browser either through a plug in or virtual machines. A site-specific browser is a browser minus the menu and controls.

With more and more services moving to the cloud and with increased internet connectivity among mobile devices RIAs are becoming increasingly popular on desktops as well as mobile devices. RIAs provide robust ways of interacting with the users with better efficiency than ever before. Consider traditional applications which generally tend to be limited to form fields, radio buttons, and check boxes. In contrast, RIAs let you perform in-line editing, drag and drop items or otherwise interact directly with the elements. Popular browser based RIAs include Flickr, Google Maps, and eBay.

Adobe describes an RIA as a lightweight application with a subset of the functionality and feature set of a desktop application while the user interface

runs in a web browser or some other application runtime. RIAs combine the strengths of both domains while liberating you from their respective constraints. In today's business environment, enterprise applications can be very complex, scalable, distributed, component-based, and mission-critical. For a while now, enterprises have had to decide between using web applications, which are often low on rich features and deploying desktop applications, which are complex and difficult to install and integrate. By combining the strengths of both web and desktop applications, RIAs add value while countering this complication.

In today's day and age the business environment is very comprehensive and truly global, with customers of content and applications being very diverse and more demanding than ever. Businesses strive for brand loyalty but it is becoming very difficult to obtain and retain extensive customer engagement. This is very essential for an organisation to succeed. RIAs tend to make these customer interactions more compelling, engaging, dynamic, and useful. We have become increasingly aware of the value of customer engagement and how important it is to build businesses.

With the onslaught of so many different types of digital devices from PCs, laptops, smart phones, and tablets cross platform user engagement has also become an important aspect of RIA development, thereby bringing in an increased need for products that are platform independent and companies are trying to answer this particular problem.

Platforms used for RIA development

With increasing demand for user friendly and highly interactive interfaces among users of web applications in large enterprises, the call for a strong development platform that covers all requirements of an enterprise level web application as well RIA became unavoidable. Businesses simply cannot ignore the benefits of enterprise RIA. Amongst the many RIA technologies available today, Flex has made a name for itself as the most powerful framework for developing RIAs. With Flex and LiveCycle Data Services (LCDS) and BlazeDS we can achieve the requirement of an enterprise level application.

A career in RIA development is becoming increasingly relevant and sought after. Also with the increasing penetration of the social web among the masses, RIAs dominate the online gaming industry as well as fields of applications such as video, sharing and recording. Development of new technologies and web standards such as HTML5 and Java Script based widget sets to provide solutions for mobile web experiences are also on

the rise. As a developer adept with RIA technologies, you can have a career ensured with top class placement in the programming and IT industry. With the increasing dependency on building and developing content for browsers, desktops and mobile platforms, this sort of talent is being sought after with great demand by all modern IT and web companies.

Frameworks are the platforms on which an RIA is built and deployed. There are a lot of different RIA frameworks. One of the biggest such platform for RIA development as already discussed is offered by Adobe which includes Adobe Flash, Flex and AIR. Another such framework by Microsoft is Silverlight. There is also an Open Source Silverlight project for the Linux operating system. Curl is an RIA framework designed for business use. Curl though does not focus on graphics and advertising, rather on applications that integrate with business data systems. Other important RIA frameworks include Google Web Toolkit, JavaFX, Mozilla Prism and OpenLazlo.

RIA development can be done through a variety of frameworks and technologies. For the back end, onecan make use of programming languages like Java, ColdFusion, PHP, Rails,.NET, etc. For the client sideyou can use established MVC frameworks including Flex/ActionScript and Ajax, some emerging ones for Silverlight, and adapted Java frameworks for JavaFX. The primary quality for a developer is the ability to be able to define the needs and capabilities of your application well if you want to determine both the back-end and the front-end application architecture well.

YouTube is one of the biggest examples of the success of RIA technologies. It is increasingly delivering full length movies and live streaming events, and doing this without any issues or interruptions often requires fine control over buffering and dynamic quality control. Kuan Yong, platforms product manager for Google's YouTube, says that despite his company's efforts to make YouTube videos run in an HTML5 player, Flash isn't going anywhere. "Flash Player addresses these needs by letting applications manage the downloading and playback of video via Actionscript in conjunction with either HTTP or the RTMP video streaming protocol," explains Yong. "The HTML5 standard itself does not address video streaming protocols, but a number of vendors and organizations are working to improve the experience of delivering video over *HTTP."* YouTube also has to offer copy protection for some videos, like YouTube Rentals. The Flash Platform's RTMPE protocol is compatible with copyright protection technology, but HTML5 is not. Flash also remains the preferred option for video embedding. In conjunc-

tion with Google's decision to bake Flash support into its Android devices, Flash doesn't appear to be going anywhere anytime soon.

In the past for web and application developers, limiting factors to be considered with the efficiency of software was the computer's processor speed and more recently the Internet bandwidth. This meant the user has to spend 10 minutes entering data and then wait two hours for calculations to run, or 10 seconds typing a URL and wait 30 seconds for the web page to load. This lead to the developers putting in most of their effort into making the processing or transfer faster and not much effort went into the efficiency of data entry fields and user engagement. Nowadays the exact opposite is true, we spend more time entering letters and numbers than we do waiting for pages to load or calculations to run. User engagement has become increasingly relevant. Radical new user interfaces are being developed for the modern day RIAs and even faster advancements are expected in the future changing how one will be interacting with digital devices.

Learning RIA technologies has never been easier. With companies trying to promote the use of their own technologies, there are innumerable resources available online either by Adobe, Microsoft or other open source communities. With the right creative and technical knowledge and prowess getting yourself hired in this area could be very easy and exciting. Through the rest of this Fast Track we will be focusing at building RIAs for the desktop as well as for mobile platforms withn the Flex 4.5 SDK and using tools like Flash Builder 4.5 .

Is HTML5 the future?

HTML5 is hailed as the future of the web and RIA development. But it could take a very long time for it to be mature enough for extensive use. HTML5 is large and complex, and if we are to trust the current projections by the people working on this particular spec like Ian Hickson of Google and David Hyatt of Apple, for all parts to be finished it would be the year 2022, some 18 years after the process started in 2004.

However, some Web sites are already using if not the complete spec but a subset of HTML5. For example, YouTube and Vimeo have already rolled out use of the video element in HTML5. The working subset of HTML5 is nowhere near the power of Flash. There are many advanced effects that are only available in Flash or Silverlight or Java. For example, Google, which is driving HTML5, relies on Flash in Google Maps for the Streetview and in Gmail for the multiple file upload capability. There are tens of thousands of

Flash games on the Web or as game apps within Facebook. This wouldn't be possible with HTML5 anytime soon.

Mobile applications

The *smart* phone

Today, there are more than 5 billion people around the world using mobile phones. No other technology is advancing at the rapid speed the mobile industry is experiencing. A new category of mobile phone is rapidly growing as well, the smart phone. A few years ago, a smart phone was something that maybe just allowed you to send an email. Today, when you think smart phone, you think email, web, games, MMS, video conferencing, basically a computer in your pocket. There are a number of companies leading the next wave of smart phone market. Google, Apple, RIM, Nokia, Microsoft, and HP all have their own operating systems and hardware. Consider this, at the end of 2009, a mobile phone running at 500 MHz with a 3 MB camera was considered screaming fast. Now, if you want something good you go for one with 1 GHz with a 1 GB of RAM, an 8 MP camera, front and rear facing cameras, proximity devices and sophisticated operating systems (OS) that rival, and in some cases exceed, what you can accomplish on your desktop.

Building Applications for Mobile Devices

With a global figure of 5 billion mobile users, it is clear that the smart phone market has massive potential for growth. So, what does it mean to develop for a smart phone? At the end of the day, there are essentially two ways you can develop for a smart phone:

- Develop directly to the software development kit (SDK)
- Develop using an intermediate technology

Each mobile device these days comes with an SDK that you can use for development. An SDK comes with the development tools, bundling tools, and emulators you need to test your code. When you need access to the latest and greatest technology, you need to use an SDK. The challenge you have with using core SDKs is that you need to use the native development language. This is different for each SDK. For instance, Apple prefers you use Objective-C whereas Google prefers you use Java.

The second way to develop mobile devices is to use an intermediate technology that allows you to build for multiple devices using only one language. An example of this is the 3D game development technology called Unity 3D. Unity uses JavaScript to let you to script your games

and then converts the JavaScript into code that will allow you to build iPhone, Android, and Windows Desktop applications. The downside to using intermediate technologies is that you are dependent on the development company to update their tools to the latest SDKs and technologies. This can be hard work as the SDKs are frequently updated. For instance, Apple has updated its iOS operating system five times in three years, and Google's Android has been updated five times in less than two years.

With all that said, it is much easier to develop using intermediate languages. You can leverage skills you already have without having to go through the learning curve of adopting a new language. Adobe has been making major leaps in developing such intermediate packages through its Flex framework. Since last summer Adobe has brought both the Flash Player and AIR (Adobe Integrated Runtime) to Google's Android. This is really big for several reasons,

- The version of Flash coming to the Android is the latest version 10 and above, not some crippled alternative.
- AIR gives you an immediate in-road into mobile device development, leveraging the tools and knowledge many might already have.
- Android runs on tablets and TVs as well as phones. The Flash Player that is now available for users of Android 2.2 and above.

There has been a lot of noise from companies such as Apple stating that Flash is a battery hog and will kill your phone's CPU. Is this true? The reality is that it is not. Tests have been conducted showing that the Flash Player on mobile devices is highly efficient and does not cause the CPU crippling results Apple is stating. The Flash Player works inside the browser in Android. You trigger the use of the Flash Player by tapping on the Flash content in the web page. For instance, you can view a Hulu.com video by tapping on the content in the page.

Flash support for mobile devices is being released by Adobe for over 20 different mobile development companies. What does this mean for you? This basically provides the user the ability to learn just one basic workflow and be able to develop for not just Android but for other platforms which have adopted Adobe's technologies like RIM's BlackBerry, Nokia, HP/Palm WebOS, and Microsoft's Windows Phones 7. The Flash Player as the plugin is also available for systems like Google TV and BlackBerry Tablet OS, LiMo and Samsung SmartTVs .

Adobe Flash Builder 4.5

It is estimated that over 131 million smart phones will be having have Flash Player installed on them by the end of this year. Adobe Flash now brings rich browser based content not only to desktops but to mobile devices as well.

Adobe just recently released its new version of the Flash builder version 4.5 alongside its Creative Suite 5.5 release. Flex 4.5 which is an open source framework now includes new and improved support for creating mobile applications across all major mobile platforms. This novel and more effective approach in mobile development lets you leverage on a single workflow, programming language and code base to easily create applications across all major mobile platforms, thus speeding up the workflow, saving time and the cost of delivery and keeping it uniform across all platforms. So this basically implies that you can make use of Flash Builder 4.5 to build great looking, high performing mobile applications more efficiently and then package them and get them ready for deployment which can be submitted to an app store or market place. You will no longer have to learn different languages and multiple tools for developing across different mobile platforms. You will be able to package apps for Android, and also you will be able to compile apps for BlackBerry Tablet OS. With Adobe's latest 4.5.1 update for Flex and Flash Builder, you can package them for iOS and well.

Around twenty other mobile development companies will be getting support for the Adobe Flash player on their platforms. Also the Flex framework keeps adding support for different devices. This implies that the concepts and development process you will be learning in this book will be useful for building applications beyond just the Android and RIM's BlackBerry tablet OS. With the June release of Flash Builder 4.5.1, you will be able to build native applications for Apple's iOS devices as well and there is definitely more to come . It is estimated that there will be over 131 million smartphones with Flash Player installed by the end of this year, Adobe Flash brings rich browser-based content to both desktops and mobile devices.

Also fortunately for mobile application developers, Adobe announced Flash Builder 4.5 recently. Flex 4.5 is an open source framework which lets you build mobile applications and provides support for them across all the popular mobile platforms. This new way of building applications enables developers to make use of a single tool, programming language and code base to create easily highly expressive applications for all major mobile platforms letting you improve performance, efficiency and reducing the time and cost of development. What does this mean? Basically Flash

Builder 4.5 can be used to build high performing mobile applications, and the ability to package them and have them ready for deployment which can be submitted to an app store or market place, thus eliminating the need for learning different languages and tools for different mobile platforms. Apart from being able to package apps for Android, you will also be able to compile apps for BlackBerry Tablet OS. Also, with the recent update for Adobe Flash Builder 4.5.1, you can develop for iOS too.

The Flex 4.5 Framework

Though Flex was originally targeted to capture the enterprise application development market with its initial releases, Adobe Flex along with the technologies surrounding it is now becoming main stream in the RIA space. Flex is a powerful application development solution for creating and delivering RIAs within the enterprise and across the web. It provides a modern, standards based language and a programming model that supports common design patterns.

With the rapid evolution of mobile computing platforms, new challenges have emerged for application developers. Much like the early days of web and desktop computing, each platform has its own development model, including programming language, framework, tools, and different deployment options. These challenges add time, cost, and complexity to delivering applications across the web, the desktop, and the many mobile device platforms. With the release of Adobe Flash Builder 4.5 and Flex 4.5, Adobe is now offering unprecedented new capabilities for building mobile applications and content. These new products offer a new mobile development paradigm, allowing the developer to use a single tool, single development framework, and a uniform codebase to build mobile applications on all the Android, BlackBerry Tablet OS and iOS platforms. Not only mobile but it offers you all of the tools that you need to build the most expressive RIAs for the web, and desktop platforms all in a single suite.

The Adobe Flash Platform which previously enabled developers to develop applications across multiple browsers and operating systems, now with the introduction of the Adobe Flex 4.5 SDK and Flash Builder 4.5 along with the availability of the Adobe AIR runtime on mobile devices, enables developers to create mobile Flex applications for touchscreen smartphones and tablets with the same ease and quality as on desktop platforms.

By giving developers a common path to creating applications for web, desktop, and multiple mobile platforms, Adobe has attempted to significantly

reduce the time and cost linked to application development. In 4.5 Flex has extended its extensive existing web and desktop component library by 21 new touch enabled, optimised, and density-aware mobile components, accelerating mobile application development. In addition to the new mobile components, Flex 4.5 adds other important capabilities and improvements for building mobile applications including kinetics and elastic bounce/pull effects for scrolling components.

Flash Builder 4.5 also adds important new mobile development workflows to help you code, debug, and optimise mobile applications. It features a new mobile project type, new design-view per-device preview, new support for multi-density authoring, per-platform application permissions editing, and a powerful new debugging workflow that allows you to debug on a physical device or on the desktop using a device emulator. You can deploy, package, and sign the required resources as a platform-specific installer file for upload to a mobile application distribution site or store.

For the rest of this book, we will be building applications for the desktop and mobile devices making use of the Flex 4.5 platform and the tools available in Flash Builder 4.5. This is a recent release from Adobe, so be sure to download the latest update before trying out the exercises in this book.

What is AIR?

The Adobe Integrated Runtime, or Air, is a cross platform runtime. Basically what it means is that it's a wrapper for rich internet applications to be installed on several Operating systems. It is freely available and downloadable from the Adobe web site just like the Flash Player. You will get rich and engaging visuals from the applications you create for Adobe Air as it includes a lot of built-in graphic capabilities. Also the applications will run on Windows, Linux and Mac machines with little or no code modification on your part. Finally your users will also enjoy a quick and simple installation through Adobe Air runtime.

You can build Air applications using HTML and JavaScript which are then rendered inside an Air application using the Open Source Webkit HTML and JavaScript engine. So, you can use a HTML editor like Dreamweaver or even plain Notepad to create Air applications. You can also build Air applications using the Adobe Flash Builder 4.5, which is exactly what we will be doing in this book. While Flex was traditionally

used to build Flex applications for the browser, the Air SDK which has been added into Flex recently gives you more features that are specific to Adobe Air.

To get your Flash apps running in AIR on all these new devices, Adobe made one major decision, you must develop your solutions using ActionScript 3.0 (AS3). You can no longer leverage the older AS1 and AS2 for the development of these apps.

MXML

Yes, when you encounter such abbreviations, it means serious business!

MXML (short for 'Macromedia eXtensibleMarkup Language') is an XML-based markup language used in Flex for defining the user interface. MXML is primarily used in combination with ActionScript to develop RIAs on the Flex platform.

MXML is primarily used for laying out interfaces while building applications and can also be used to deploy business logic and internet application behaviours. It usually might constitute within it chunks of ActionScript code, either when creating the body of an event handler function, or with data binding where the curly braces ({}) syntax is used.

MXML on a Flex Server is dynamically compiled into a standard binary SWF file as with most Flash files. However, the Adobe Flash Builder IDE and the free Flex SDK also allow you to compile MXML into SWF files without the use of a Flex Server. Therefore, MXML can be considered as a proprietary standard due to the fact that it is tightly integrated with Adobe technologies only. Hence no formally published translators exist for converting an MXML document to another user interface language such as UIML, XUL, XForms, XAML, or SVG. None the less there are a few third party vendor plug-ins that are available for Flash Builder 4.5 which let you generate a result other than in an SWF file.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009" xmlns:s="lib
               xmlns:mx="library://ns.adobe.com/flex/mx" xmlns:d="ht
               minWidth="1024" minHeight="768">
    <!-- Non-ASDoc Comment -->
    <fx:Script>
        <![CDATA[
            //ActionScript
        ]]>
    </fx:Script>
    <fx:DesignLayer d:id="3" d:userLabel="shapes">
        <s:Ellipse y="105.5" x="441.5" height="319" width="134" id="
            <s:fill>
                <s:SolidColor color="0xFFFFFF"/>
            </s:fill>
        </s:Ellipse>
    </fx:DesignLayer>

    <!--- ASDoc comment for myButton -->
    <s:Button id="myButton"/>
</s:Application>
```

Sample MXML code

Syntax

Here's the typical Hello World example in MXML:

- `<?xml version="1.0" encoding="utf-8"?>`
- `<mx:Applicationxmlns:mx=" learn.adobe.com/wiki/display/Flex/MXML"`
- `layout="absolute" backgroundGradientColors="[#000000, #666666]">`
- `<mx:Label text="Hello World!" fontSize="40" letterSpacing="2"verticalCenter="0" horizontalCenter="0" >`
- `<mx:filters>`
- `<mx:GlowFiltercolor="#dddddd"/>`
- `</mx:filters>`
- `</mx:Label>`
- `</mx:Application>`

Creating a UI with MXML

Graphics using MXML and FXG

One of the biggest features of the Flex 4.5 SDK is its ability to define the visual controls' appearance with programmable skins. Skinning and the concept of programming a component's visual appearance, isn't new to Flex. In its previous versions, you could create your own skin as a newActionScript class. All you need to do is override certain methods that are called during runtime from the Flex framework and use the Flash drawing application programming interface (API) to declare vector graphics. With this, you can create an entire visual presentation without ever using software such as Adobe Photoshop and Illustrator to draw interfaces.

Unfortunately, this method is way too slow and cumbersome. Also, Photoshop can't interpret ActionScript code to let you preview results, and effectively you'd have to do some serious planning on how to code the image you intend to create. To ease this, Adobe introduced the FXG language to help solve this particular programming hassle. FXG, or Flash XML Graphics, is a graphics interchange file format based on XML that lets you define low-level vector graphics and specific attention is paid to how Flash Player renders graphics. Starting from Flex 4, you can declare FXG graphics either as part of an MXML file or by incorporating FXG files exported from Creative Suite products as embedded graphics.

Declaring vector graphics in MXML

FXG is basically an XMLbased declarative format that enables you to either declare vector graphics within an MXML file or exchange graphical declarations between software applications that support the format that is using FXG files.There are two versions of the FXG specification: versions 1 and 2. Starting with Creative Suite 4, Adobe's graphical authoring tools (Illustrator, Photoshop, and Fireworks) were able to import and export FXG files. In Creative Suite 4 (CS4), these software products work with FXG version 1, while the CS5 versions of these applications and the Flex 4.5 SDK supports both versions 1 and 2.

Note that you can define a vector graphic in text files with an FXG extension, or as an MXML-based declaration of ActionScript objects that are based on a set of classes. In either case, the graphics are rendered by Flash Player at runtime and are dependent on Flash Player's graphical rendering capabilities.

Drawing lines and shapes

In an FXG file, you can simply draw a shape by declaring an element with its name and then setting its required attributes.The FXG specification has the following elements defined

- `<Graphic>`
- `<Ellipse>`
- `<Line>`
- `<Path>`
- `<Rect>`

To accomplish the same thing in MXML, the Flex 4.5 SDK includes a package named spark.primitives. This package contains the following ActionScript classes that you can use to declare vector graphics:

- **Ellipse** - Draws an elliptical shape. If its height and width are identical, it draws a circle.
- **Line** - Draws a straight line from one set of coordinates to another.
- **Path** - Draws a shape based on a set of drawing commands, creating multiple shape segments.
- **Rect** - Draws a rectangular shape. If its height and width are identical, it draws a square.

Drawing lines

The Line class draws a line on the screen. As with all such primitives, it must be placed within a Spark application or component. Its width and height properties determine its horizontal and vertical length, while the stroke property determines its colour and style. The stroke property must be set to an instance of a class that implements the IStroke interface. Examples of such classes in the Flex 4.5 SDK include GradientStroke and SolidColorStroke. The following code draws a simple horizontal line. The stroke property is set to an instance of SolidColorStroke with a color of black (#OOOOOO) and a weight of 2 pixels:

```
<s:Line width="700">
<s:stroke>
<s:SolidColorStrokecolor="#000000" weight="2"/>
</s:stroke>
</s:Line>
```

Using gradient strokes

As with the fill property used with shapes that are drawn with the Rect, Ellipse, and Path classes, the stroke property can be set to a gradient of two or more colours with the GradientStroke class or with one of its Subclasses, LinearGradientStroke and RadialGradientStroke.

The gradient stroke classes support a property named entries that's set to an array of two or more instances of the GradientEntry class. The following code draws a horizontal line that's 10 pixels wide and has five entries:

```
<s:Line width="7 00">
<s:stroke>
<s:LinearGradientStroke weight="10">
<s:entries>
```

```
<s:GradientEntrycolor="#000000"/>
<s:GradientEntrycolor="#ffffff"/>
<s:GradientEntrycolor="#000000"/>
<s:GradientEntrycolor="#ffffff"/>
<s:GradientEntrycolor="#000000"/>
</s:entries>
</s:LinearGradientStroke>
</s:stroke>
</s:Line>
```

The resulting line will have alternating black and white colours.

Drawing rectangular and elliptical shapes

The two most commonly used primitive vector graphic classes are Rect and Ellipse. Respectively, they render rectangular and elliptical shapes on the screen. Both support fill and stroke properties, which enable you to define the outer border and inner fill of the shape you're drawing.Each shape's fill property can be set to an instance of a class that implements the IFill interface.

A simple rectangle with a light gray fill and a black stroke.It is defined as an instance of the Rect class

And, as described in the previous section on the Line class, each shape's stroke property is set to an instance of a class that implements the IStroke interface. The following MXML code defines a rectangle with dimensions of 400 pixels width by 300 pixels height. The rectangle's outer border is a solid black line with a weight of 2 pixels, and the fill is a solid light gray:

```
<s:Rect width="400" height="300"
horizontalCenter="0" verticalCenter="0" >
<s:fill>
<s:SolidColorcolor="#EEEEEE"/>
</s:fill>
<s:stroke>
<s:SolidColorStrokecolor="#000000" weight="2"/>
</s:stroke>
</s:Rect>
```

Drawing arbitrary shapes with the path class

The Path class enables you to declare any shape based on a set of commands that replicate the features of the Flash drawing API. Its data property is a string that executes cursor placement and drawing operations. The data string alternates commands and sets of numeric values. Each command is notated with a single alphabetical character, as follows:

- **C -**Draw a cubic Bezier curve. The first two values are the first set of control coordinates, the second two values are the second set of control coordinates, and the last two values are the drawing destination.
- **H -** Draw a horizontal line from the current cursor coordinate to a new X coordinate.
- **L -** Draw a line from the current cursor position to a set of coordinates. For example, the command L 100 100 causes a line to be drawn from the current cursor position to X and Y coordinates of 100 and 100.

- **M -** Move the cursor to a set of coordinates. For example, the command M 50 100 causes the cursor to be placed at an X coordinate of 50 and a Y coordinate of 100.
- **Q -** Draw a quadratic Bezier curve. The first two values are the control coordinates, and the last two are the drawing destination.
- **V -** Draw a vertical line from the current cursor coordinate to a new Y coordinate.
- **Z -** Close the path.

The following simple Path object draws a horizontal line starting at X and Y positions of 100, and then draws a horizontal line to an X position of 500. The colour and weight of the path are determined by its stroke property:

```
<s:Path data="M 100 100 H 500 Z">
<s:stroke>
<s:SolidColorStrokecolor="black" weight="5"/>
</s:stroke>
</s:Path>
```

More complex Path objects can be drawn with more commands, and multiple shapes and lines can be drawn in the same Path object by moving

the cursor and initiating new draw commands. The following object draws a curved arrow using a series of lines and cubic Bezier curves:

```
<s:Path data="M 20 0
      C 50 0 50 35 20 35
      L 15 35 L 15 45
      L 0 32 L 15 19
      L 15 29 L 20 29
      C 44 29 44 6 20 6">
<s:stroke>
<s:SolidColorStrokecolor="0x888888"/>
</s:stroke>
<s:fill>
<s:LinearGradient rotation="90">
<s:GradientEntrycolor="0x000000" alpha="0.8"/>
<s:GradientEntrycolor="0xFFFFFF" alpha="0.8"/>
</s:LinearGradient>
</s:fill>
</s:Path>
```

Adding visual effects

The primitive vector graphic classes support many properties that enable you to modify their appearance. Examples include gradient fills and strokes, drop shadows and other filters, and scaling, or resizing, of vector graphics. In this section we look at some of the most commonly used effects from which you can choose.

Using gradient fills

The fill property that's supported by the Rect, Ellipse, and Path classes can be set to a solid color with the SolidColor class or to a gradient with either RadialGradient orLinearGradient. Each does exactly what its name implies:

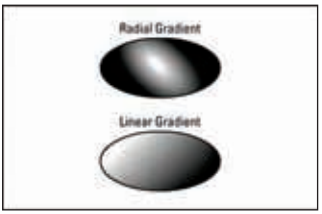
- **LinearGradient -** Defines a change in colours in a linear using calculation from one coordinate to another. By default the gradient is calculated from left to right, but it can be adjusted by changing the gradient class's direction property; for example, to change the gradient to go from top to bottom, change the LinearGradient object's direction to 90.
- **RadialGradient -** Defines a change in colours starting from the certain point in an object (by default its centre) and radiating outward to its borders. You can set the focalPoint Ratio and rotation properties to

change the point from which the gradient radiates.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Applicationxmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
      @namespace s "library://ns.adobe.com/flex/spark";
s|Label{ font-size:14; font-weight:bold; }
</fx:Style>
<s:layout>
<s:VerticalLayout gap="20" horizontalAlign="center" pad-
dingTop="20"/>
</s:layout>
<s:Label text="Radial Gradient"/>
<s:Ellipse width="200" height="100"
horizontalCenter="0" verticalCenter="0" >
<s:fill>
<s:RadialGradientfocalPointRatio="-.1" rotation="45">
<s:entries>
<s:GradientEntrycolor="#FFFFFF"/>
<s:GradientEntrycolor="#000000"/>
</s:entries>
</s:RadialGradient>
</s:fill>
<s:stroke>
<s:SolidColorStrokecolor="#000000" weight="2"/>
</s:stroke>
</s:Ellipse>
<s:Label text="Linear Gradient"/>
<s:Ellipse width="200" height="100"
horizontalCenter="0" verticalCenter="0">
<s:fill>
<s:LinearGradient rotation="45">
<s:entries>
<s:GradientEntrycolor="#FFFFFF"/>
<s:GradientEntrycolor="#000000"/>
</s:entries>
</s:LinearGradient>
</s:fill>
<s:stroke>
<s:SolidColorStrokecolor="#000000" weight="2"/>
```

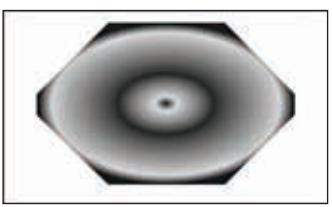
```
</s:stroke>
</s:Ellipse>
</s:Application>
```

The fill property contains an array of GradientEntry objects. Each can be assigned a ratio value from 0 to 1 that determines where in the graphical element the colour change occurs. The followingPath object uses a RadialGradient with many instances of the GradientEntry class. They alternate colours and determine where the colours change in the graphical element with their ratio values:



Visual result: two Ellipse objects with different types of gradient fills.These ellipse objects use, respectively, a linear and a radial gradient fill

```
<s:Path data="M 0 40 L 40 0 L 105 0 L 144 40 L 144 51L 105 90 L 40 90 L 0 51 L 0 40 Z ">
<s:fill>
<s:RadialGradient>
<s:GradientEntrycolor="#333333" ratio="0" alpha="1"/>
<s:GradientEntrycolor="#CCCCCC" ratio="0.09" alpha="1"/>
<s:GradientEntrycolor="#333333" ratio="0.37" alpha="1"/>
<s:GradientEntrycolor="#CCCCCC" ratio="0.89" alpha="1"/>
<s:GradientEntrycolor="#333333" ratio="0.99" alpha="1"/>
</s:RadialGradient>
</s:fill>
</s:Path>
```



The result: a symmetrical polygon with a series of concentric radial gradients.A symmetrical polygon with a radial gradient fill defined with primitive vector graphic classes declared in MXML

Reusing graphic elements with <fx:Library> and <fx:Definition>

The Flex SDK now has a new MXML element named <fx:Library> that you use to define graphic elements that can then be reused anywhere in the same MXML document. Within the <fx:Library>element, you define one or more <fx:Definition> element. Each <fx:Definition> describes a graphic

element. Note, the <fx:Library> tag must be placed as the first child element within the MXML document's root element. If you place any other tags before it, a compiler error is generated. The application CurvedArrow below defines a single vector graphic as a library definition. In the body of the application, there are three instances of the graphic. Each sets its scaleX and scaleY properties to a different value. The resulting application displays three instances of the arrow. Even though they're different sizes, they all show clean curves and smooth gradient fills.

A library definition shape used multiple times

```
<?xml version="1.0" encoding="utf-8"?>
<s:Applicationxmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark">
<fx:Library>
<fx:Definition name="CurvedArrow">
<s:Path data="M 20 0
                      C 50 0 50 35 20 35
                      L 15 35 L 15 45
                      L 0 32 L 15 19
                      L 15 29 L 20 29
                      C 44 29 44 6 20 6">
<s:stroke>
<s:SolidColorStrokecolor="0x888888"/>
</s:stroke>
<s:fill>
<s:LinearGradient rotation="90">
<s:GradientEntrycolor="0x000000" alpha="0.8"/>
<s:GradientEntrycolor="0xFFFFFF" alpha="0.8"/>
</s:LinearGradient>
</s:fill>
</s:Path>
</fx:Definition>
</fx:Library>
<s:HGrouphorizontalCenter="0" top="20">
<fx:CurvedArrowscaleX="1" scaleY="1"/>
<fx:CurvedArrowscaleX="2" scaleY="2"/>
<fx:CurvedArrowscaleX="4" scaleY="4"/>
</s:HGroup>
</s:Application>
```

The <fx:Library> MXML tag is new in Flex. You use it to define reusable graphic elements. Each reusable graphical element is defined in

an <fx:Definition> class with aname property. The definition becomes an ActionScript class at compilation time. You can then declare instances of the reusable graphic element by referring to its name as an MXMLtag with the fx prefix. For example, in the code, the <fx:Library> tag is a direct child of <s:Application>. It contains this definition:

```
<fx:Definition name="CurvedArrow">
<s:Path.../>
</fx:Definition>
```

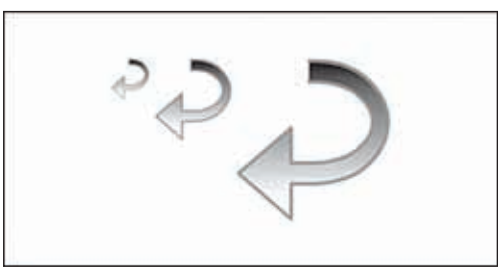
The resulting CurvedArrow graphical element can then be used anywhere in the visual content of the document in which the <fx:Library> tag is declared.

Scaling graphic elements

An MXML graphic is rendered as a vector graphic; this means that its rendering is calculated mathematically rather than as a set of pixels (as is the case with bitmap graphics). As a result, you can increase or decrease the size of the graphic (a process known as scaling) without disturbing the graphic's resolution. When you do the same thing with a bitmap graphic, it's often pixelated, showing jagged edges. With vector graphics, the rendering stays clean and attractive.

The graphic that's defined in the <fx:Library> section is rendered three times, each time with a different value for the object's scaleY and scaleY properties:

```
<fx:CurvedArrowscaleX="1" scaleY="1"/>
<fx:CurvedArrowscaleX="2" scaleY="2"/>
<fx:CurvedArrowscaleX="4" scaleY="4"/>
```



The result, with the three versions of the arrow graphic displayed side by side. Notice that the graphic's curves are smooth regardless of the scale. This is a benefit of vector graphics: because their presentation is calculated mathematically, they can adjust to whatever scale your application needs. A graphic element displayed in three scales

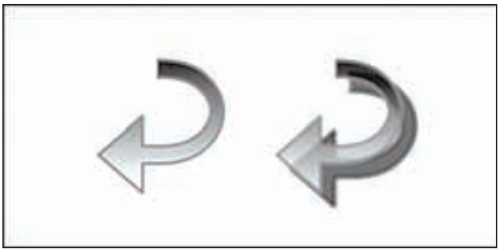
Applying filters

Each of the primitive vector graphic classes supports the filters property, an array of objects derived from one of the classes in the spark.filters package. The following filter classes are included in the Flex 4.5 SDK:

- BevelFilter
- BlurFilter
- ColorMatrixFilter
- ConvolutionFilter
- DisplacementMapFilter
- DropShadowFilter
- GlowFilter
- GradientBevelFilter
- GradientFilter
- GradientGlowFilter
- ShaderFilter

Each filter class supports a set of properties that modify its effect on the graphical element. For example, the following Path object adds two filters, a drop shadow, and a blur:

```
<s:Path data="M 20 0
      C 50 0 50 35 20 35
      L 15 35 L 15 45
      L 0 32 L 15 19
      L 15 29 L 20 29
      C 44 29 44 6 20 6">
<s:stroke>
<s:SolidColorStrokecolor="0x888888"/>
</s:stroke>
<s:fill>
<s:LinearGradient rotation="90">
<s:GradientEntrycolor="0x000000" alpha="0.8"/>
<s:GradientEntrycolor="0xFFFFFF" alpha="0.8"/>
</s:LinearGradient>
</s:fill>
<s:filters>
<s:DropShadowFilter distance="20" color="#333333"/>
<s:BlurFilter/>
</s:filters>
</s:Path>
```



Two versions of the resulting graphical element side by side. The first is the graphical element without the filters; the second shows the changes implemented by the filters.A graphical element before and after adding filters

Also keep in mind that you can make use of Adobe Flash filters to apply special effects like different styles to the Flex components for example Labels and Text. You can apply these filters to any visual component in Flex that is derived from the UIComponent. Note that Filters through are not styles because you will not be able to apply them using a style sheet or the setStyle() method. The result of a filter, none the less is often thought of as a style.

ACTION SCRIPT 3

ActionScript 3.0 is an object-oriented language developed primarily by Adobe for creating applications or multimedia content which can be run in Flash client runtimes like Flash Player and AIR

ActionScript 3.0

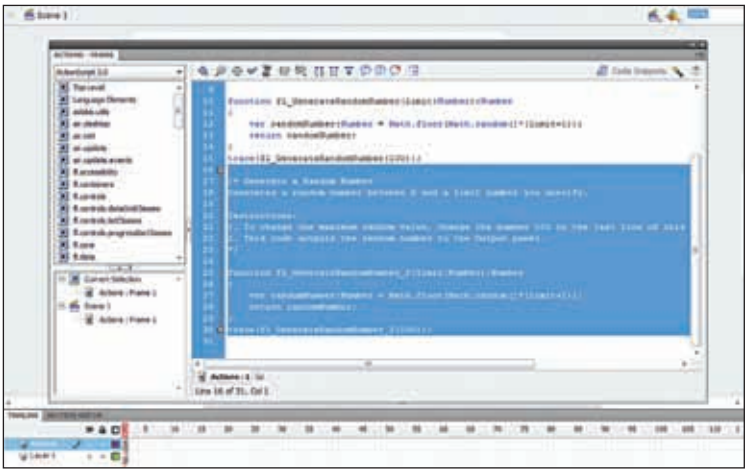
ActionScript 3.0, at its core, is a language based on the ECMAScript4 just like the very popular language used in web browsers JavaScript. ECMAScript 4 basically dictates all of ActionScript's basic syntax.

AS3 has very good support for common object-oriented constructs such as classes, objects, and interfaces. It also has Runtime type checking and Optional compile time type checking. It includes dynamic features such as runtime creation of new constructor functions and variables. It offers direct support for XML as a built-in data type. All Flash client runtimes that support ActionScript 3.0 share the features of the core language in common.

There are two primary routes you can take while handling ActionScript. The first method which is more commonly used by earlier Flash developers is to simply put in your ActionScript directly into the timeline. In this method, having a separate layer in your timeline that is primarily dedicated just for working with ActionScript is considered a good practice. Adobe recommends that ActionScriptlabel be labelled as 'Actions.' By locking the Actions layer in the timeline you can save yourself the trouble

ActionScript 3 Code Snippets in Flash CS5

of accidentally adding movie clips into it. The other method of coding in ActionScript and inserting it into your Flash files is by using a Class file. While inserting a Class file you can now specify both public and private classes. Using the keyword 'Private' will restrict the access of the attribute you defined to just that class and it will not be shared with other classes. This is very useful when you develop instructions that need to be executed privately in a closed secure environment.



Generating Random Numbers Action

Working with classes

A common design pattern in object-oriented programs separates design, programming, and data elements. Flash CS5 adds this functionality with the inclusion of classes. A class is a packaged document that you can use to explain how UI components,business logic, and data elements will interact.

A class is a separate ActionScript file that is associated with the main Flash file and movie clips. You can use Flash CS5 as the class file editor or your favourite text editor such as Eclipse,Notepad, or TextEdit. A Class file is only a text file. It is very easy to create entire Flash movies using just Class files and not even add any content into a traditional timeline.

These steps will show you how to create a simple Class file for your Flash movies:

1. Create a new AS3 file. Save the file and name it “helloWorld.fla”.
2. In the new, blank helloWorld.fla file open the Properties panel.
3. Expand the Publish setting. You will see a Class field. To theright-hand side of the Class field is a small pencil icon. Selectthe icon. A new window will open asking you if you want to createa new class. Create a new class and call it helloClass.
4. A new ActionScript file will open. Notice that the file is now labeledhelloClass. The class is a default, blank class with the ActionScript.

```
package {
importflash.display.MovieClip;
public class helloClass extends MovieClip {
public function helloClass() {
//constructor code
}
}
}
```

5. Remove the line that says //constructor code and replace itwith: trace ("Hello, World");
6. Save your Class file.

Return to the helloWorld.fla file and test the movie. The result should be the words "Hello, World" posted to the Output panel.

Classes provide you a way in which you can create public and private class elements. The difference between the two is related to how you use the data. For instance, a public property can be shared throughout your whole Flash movie. A private property can only be used within the class in which it is defined.

Using namespaces

Namespaces are ways in which you can define the visibility of properties you're creating. This is commonly used in XML when you're importing documents using a URI indicator.

The following example is built using a class called NamespaceExample. The role of this class is to pull in an XMLdocument and step through the formatting of the code. Using namespace, you can instruct Flash where to find a definition of the document type you're using. In this case, an RSS formatted-document type.

1. Create a new ActionScript 3.0 movie. Create the classNamespaceExample.
2. Create a simple RSS formatted XML document. You can use the following formatted RSS document:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://purl.org/rss/1.0/"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel rdf:about="http://www.xml.com/cs/xml/
query/q/19">
<title>This is an RSS feed</title>
<link>http://www.bbc.co.uk/</link>
<description>This is a test RSS document.
</description>
```

```
<language>en-us</language>
<items>
<rdf:Seq>
<rdf:lirdf:resource="http://www.bbc.co.uk/"/>
</rdf:Seq>
</items>
</channel>
<item rdf:about="http://news.bbc.co.uk/">
<title>BBC News Center</title>
<link>http://news.bbc.co.uk</link>
<description>Welcome to the BBC News Center</description>
<dc:creator>BBC</dc:creator>
<dc:date>2010-02-12</dc:date>
</item>
<item rdf:about="http://www.bbc.co.uk/radio">
<title>BBC Radio Center</title>
<link>http://www.bbc.co.uk/radio</link>
<description>Welcome to the BBC Radio Center
</description>
<dc:creator>BBC</dc:creator>
<dc:date>2010-02-12</dc:date>
</item></rdf:RDF>
```

3. Open the NamespaceExample class. Start by defining the package with a public class called NamespaceExample that will extend the functionality of the Sprite object:

```
package
{
importflash.display.Sprite;
public class NamespaceExample extends Sprite
```

4. Insert the namespace reference that describes how to use RSSXML:

```
{
privatevarrss:Namespace = new Namespace("http://purl.org/
rss/1.0/");
privatevarrdf:Namespace = new Namespace("http://www.
w3.org/1999/02/22-rdf-syntax-ns#");
privatevardc:Namespace = new Namespace("http://purl.org/dc/
elements/1.1/");
public function NamespaceExample()
```

5. RSS has several standard XML types. You are going to extract the following: title, creator, date, link, and description. Each of these items will

be formatted in accordance to the namespace called RSS. You will see in the third line of the ActionScript that you reference the RSS namespace.

```
private function parseRSS(rssXML:XML):Array
{
default xml namespace = rss;
varitems:XMLList = rssXML.item;
vararr:Array = new Array();
varlen:uint = items.length();
for (var i:uint; i <len; i++)
{
arr.push({title:items[i].title,
creator:items[i].dc::creator, date:items[i].dc::date,
link:items[i].link, description:items[i].description});
}
returnarr;
}
```

6. The final step is to add a Public function that will use the RSS namespace and send the content to the Output panel:

```
public function NamespaceExample()
{
varmyXML:XML = getRSS();
varrssItems:Array = parseRSS(myXML);
varlen:uint = rssItems.length;
for (var i:uint; i <len; i++)
{
trace(rssItems[i].title);
trace(rssItems[i].creator);
trace(rssItems[i].date);
trace(rssItems[i].link);
trace(rssItems[i].description);
}
}
```

Run your Flash movie to see the RSS feed results sent to your Output panel. Namespaces are an effective way to manage your control over XML-data. As with all core classes in Flash, you can extend the namespaceto use it in conjunction with other objects and data types.

Controlling data (E4X)

There are several key ways to control data in AS3. They include arrays,

shared objects, and XML. An array is the first method you're likely to use in your ActionScript code. The role of an array is to create a list of data types in your code. For example, you may want to list the colours red, green, blue, and orange. To do this you need to define a new variable with the data type of Array:

```
varcolorArray:Array = new Array("red", "green",
"blue", "orange");
```

You can see in this script that a set of four items have been inserted into the array. You can access the data in the array with the following trace statement:

```
trace (colorArray);
```

The "push" property will allow you to add a new item into your array:

```
colorArray.push("purple");
```

To remove the last item of an array you can use the Pop property.

```
colorArray.pop();
```

What you will find is that arrays are great for managing simple lists. Additional properties allow you to remove specific values, to count the number of values you have, and to sort your lists. For more complex data you will want to leverage the Local DataStorage or XML.

Using flash cookies

The Flash Player can store data locally in very much the same way that a cookie can be stored in a web browser. Flash does not call them cookies, but Shared Objects. An example of a Shared Object in AS3 is:

```
varmySO:SharedObject = SharedObject.
getLocal("myFlashCookie");
mySO.data.now = new Date().time;
trace(mySO.data.now);
```

The Shared Object is declared and given a name where it will be stored on the local computer. You can now effectively target data to this space that can be accessed if this computer comes back to this page at a later date.

Manipulating XML with E4X

Flash has supported XML in one fashion or another since Flash 5. AS3 now supports the ECMA XML standard called E4X. You can now more easily step through your XML documents. The following will demonstrate how you can import an XML documents into your Flash movie as a data type.

1. Before you can import an XML document, you need to have one you can use. You can copy the following code and save it as an XML document labelled "colors.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<pallette>
<color>Orange</color>
<color>Red</color>
<color>Yellow</color>
</pallette>
```

2. Create a new Flash AS3 movie and save it to the same folder as the XML document.
3. Create a new object to manage the XML:

```
varmyXml:XML;
```

4. Now create a new URLLoader file that will load the XML file:

```
varxmlLoader = new URLLoader();
xmlLoader.addEventListener(Event.COMPLETE,onXMLLoaded);
xmlLoader.load(new URLRequest("colors.xml"));
```

5. At this point you have loaded the XML successfully intoFlash. You can test this by adding the following function to trace the contents of the XML document into your Outputwindow.

```
functiononXMLLoaded(e:Event):void{
myXml = new XML(e.target.data);
trace(myXml);
}
```

6. The result should look just like your XML document.
7. You can now easily pull out a specific value. For instance, add the following to the onXMLLoaded function to extract the third value in the XML file:

```
trace(myXml..color[2]);
```

The double dots after the variable myXML allow you to step to the second value of your XML document. All of this is so much easier to accomplish with E4X than with the AS2 version.

Regular expressions

Patterns are everywhere as you develop your code. This is clearly seen with the use of regular expressions, a method for describing the pattern of data you are looking to use. Using regular expressions, you can now easily format form fields to correctly capture date, ZIP, or credit card numbers. You can use a simple pattern with a string variable to validate the data:

```
varmyColor = "Orange";
```

Now create a new Regular Expression that is looking for a simple pattern. In this instance, the pattern is that the myColor stringvalue must start with an O.

```
varcolorRegExp:RegExp = /O/;
```

You can write a trace script to test your movie:

```
trace(colorRegExp.test( myColor ) );
```

The value in the Output panel is True. Let's extend what you can do with Regular Expressions by adding a pattern that looks for an email address. Start by adding anew e-mail string with a valid email address:

```
varemail:String = "test@thinkdigit.com";
```

Next, create a new Regular Expression that is looking for a pattern-structure in your email:

```
varemailRegExp:RegExp = /^([a-zA-Z0-9 _ -]+)@([a-zA-Z0-9.-]+)\.([a-zA-Z]{2,4})$/i;
```

The pattern is looking for a combination of alpha-numeric special character formats separated by an @ sign and suffix ".".Add the following trace statement to see whether or not the pattern works:

```
trace( "Is this email valid? " + emailRegExp.test( email ) )
```

Test the movie and you will get the following response in the Output panel:

```
Is this email valid? True
```

Change the email address to just "Digit" a pattern that does not match the Regular Expression. When you test the movie you will see that the Regular Expression returns a false response.

Controlling text

In many ways you do not need to work on the stage at all when using AS3. All visual objects can be programmatically created. The easiest way to see this is in using the Text object to create dynamic text fields on the stage.

1. To create a dynamic text field, create a new AS3 file with an associated class called text.
2. The Actions panel will open showing you the text Class file. Add the libraries to be imported into your file:

```
importflash.display.Sprite;
importflash.text.TextField;
importflash.text.TextFieldAutoSize;
importflash.text.TextFormat;
```

3. Now you need to insert a private variable that will be used to define the

dynamic text:

```
privatevarmyTextField:TextField;
```

4. The following creates a basic string you can insert into your text field:

```
privatevarsomeText:String = "Hello world.";
```

5. A private function is used to define the physical position of the text field on the screen. You first need to declare the text field as a new object; then you can use the X and Y properties to place the text on the screen:

```
private function configuretext():void
{
myTextField = new TextField();
myTextField.y = 200;
myTextField.x = 100;
```

6. A TextFormat object is used to format the visual properties of the text. For instance, the following TextFormat object sets the font to "_sans", the colour black, and font size 15:

```
varformat:TextFormat = new TextFormat();
format.font = "_sans";
format.color = 0x000000;
format.size = 15;
myTextField.defaultTextFormat = format;
addChild(myTextField);
```

7. The final two public functions tie the text string to the new formatted text field:

```
public function text()
{
configuretext();
setValueOfTextField(someText);}
public function setValueOfTextField(str:String):void
{
myTextField.text = str;
}
```

8. Test your movie and you will see that you have a text string added to your screen.

So why would you go through the hard work of adding a scripted text field to the screen when you can do the same thing with the Flash text object with no scripting? The reason is that there may be times when you want to dynamically create text fields and the TextField object gives you this option.

Drawing with the shape class

As with the text object, you can create images dynamically in AS3. There are several different types of image you can create,including traditional movie clips and graphics. You can now also create a new type of image called a Sprite. Essentially, a Sprite is the same as a movie clip with the exception that it does not contain timeline functionality.

Sprites can be created by invoking the new Sprite Object Class and then adding properties to the object. The following steps will add a new square-shaped Sprite to the stage:

1. Add the following ActionScript to create a new Sprite labelled "myFirst-Sprite."

```
varmyFirstSprite:Sprite = new Sprite();
addChild(myFirstSprite);
```

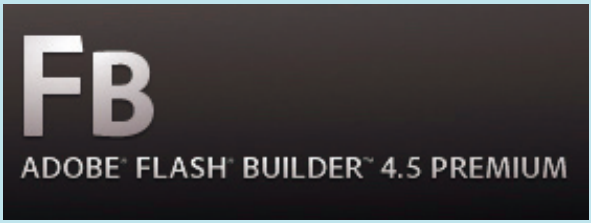
2. Format the size, fill/outline color, and position of the Sprite:

```
myFirstSprite.graphics.lineStyle(3,0xFF6600);
myFirstSprite.graphics.beginFill(0xFF0000);
myFirstSprite.graphics.drawRect(0,0,100,100);
myFirstSprite.graphics.endFill();
```

3. Now you can test the movie and see your rectangle on the screen.
4. Of course, this being ActionScript you can now add interactivity to your new Sprite. The following ActionScript will apply a fade-in transition effect to your new Sprite.

```
myFirstSprite.addEventListener(Event.ENTER _
FRAME,fadeInSprite);
myFirstSprite.alpha = 0;
functionfadeInSprite(event:Event)
{
myFirstSprite.alpha += 0.01;
if(myFirstSprite.alpha>= 1)
{
myFirstSprite.removeEventListener(Event.ENTER _ FRAME,
fadeInSprite);
}
}
```

You can do a lot with ActionScript constructed images. Working with all the different objects available to you in AS3, you have almost no limits to what you can create using Flash.



FLASH BUILDER

Adobe's Integrated Development Environment, Flash Builder lets you build applications for multiple platforms

Adobe Flash Builder 4.5, formerly Adobe Flex Builder, is an Integrated Development Environment (IDE) from Adobe Systems Inc. for ActionScript and Flex development. It's built on top of Eclipse, an open-source, extensible development platform and a popular Java IDE. Because of this, Flash Builder 4.5 inherits an impressive list of capabilities and is a familiar tool for many developers. With the latest version now Adobe includes support for building mobile applications as well as many developer productivity features.

Flash Builder 4.5 is Adobe's preferred development tool for building applications with the Flex 4.5 SDK. Flash Builder is available for both Windows and Mac OS X. Although you can develop and deploy Flex applications to the Web or the desktop with the free Flex SDK, Flash Builder is a worthwhile investment that can increase developer productivity, reduce bugs, speed up coding, and generally make the process of developing a Flex application much more enjoyable.

Flash Builder was named Flex Builder in releases prior to version 4. It's been renamed by Adobe to position it as the preferred programmer's editor for all Flash Player programming, whether by Flash or Flex developers. You can get Flash Builder from Adobe as a free limited-time trial, or you can purchase a license. Flash Builder can be installed either as a stand-alone installation that includes everything you need or as a plug-in on top of an existing installation of Eclipse. Regardless of which installation option you select, Flash Builder runs as a plug-in, or an integrated component, of another software product called Eclipse. So, before installing Flash Builder, it's important to understand the nature of Eclipse first. To install Flash Builder as a plug-in on top of your existing Eclipse installation, use the appropriate installation application for your operating system. The Flash Builder plug-in installation is self-explanatory. Just follow the prompts to complete the installation, and then start up your copy of Eclipse. Along the way, you'll be asked whether you want to install a full copy of Eclipse as a preview or install the plug-in into an existing Eclipse installation.

Using Flash Builder

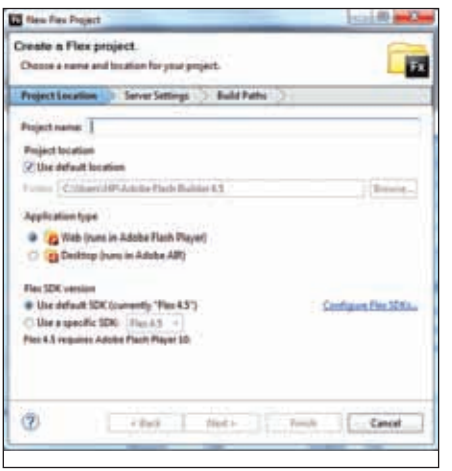
Flash Builder has a common set of tools that you use to create and test Flex applications, whether it's installed with the stand-alone or plug-in configuration. In this section, we describe the most common tasks related to Flex application development: creating a Flex project and finding Help resources.

Creating a Flex project

An Eclipse project is a collection of applications and their associated resources. When using Flash Builder, you should create your project as a resource known as a Flex project. In addition to standard Eclipse project settings, a Flex project contains many configuration options that are designed specifically for Flex developers. Choose File > New > Flex Project from the Flash Builder menu to create a new Flex project.

The startup screen of Flash Builder in the default Flash perspective

Project location can be anywhere on your disk. The default location is a folder named just like the project, placed under the workspace folder, but you don't have to put it there. This is where the project configuration and primary source-code files, and possibly compiled applications, are stored. Application type is set to either Web application or Desktop application. Selecting Web application causes the application to be delivered through the browser and run in Flash Player. Selecting Desktop application creates an application that installs for use with Adobe AIR and runs as a native application on the user's desktop.



New Flex Project Window

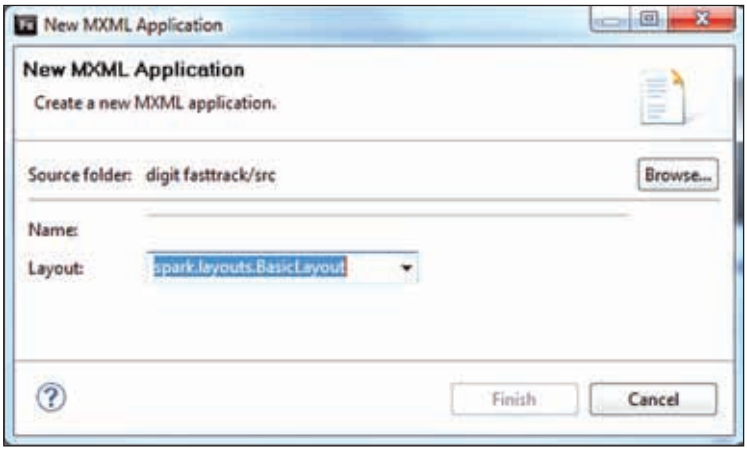
Flash Builder does not enable you to create a single Flex project whose applications can be deployed on either Flash Player or AIR. Each project must specify one and only one of these deployment options. Flash Builder can share resources between multiple projects so that each project is created as a shell for a particular deployment option, and the bulk of an application's resources are maintained in a third project known as a Flex Library Project, which compiles to a shareable SWC file. In Flash Builder 4.5, you can convert a project that was originally created for the Web to a desktop application that's deployed with Adobe AIR. Right-click on the project in the Project Navigator view and choose Add. Change Project Type > Convert to Desktop/Adobe AIR Project. You'll need to make any required code changes to the project, such as changing the root element of the main application file to use the Windowed Application component, but you won't need to recreate the project from scratch.

The Flex SDK version is set to the default SDK (currently "Flex 4.5") or a specific version of the SDK. Flex 4.5 projects use the new Spark components but can also use the older Halo components. They also implement new view state management syntax and other language features of the newer SDK. Flex 3.4 and older projects don't have access to the new Spark components and use the older view state management syntax.

ASP.NET, ColdFusion, J2EE (also known as Java EE or simply JEE) and PHP are the server technologies directly supported by Flash Builder. When you select J2EE as your server technology, you can then select either Adobe LiveCycle Data Services ES or the open-source product BlazeDS. When you select ColdFusion as your server technology, you're also prompted to select LiveCycle Data Services ES, BlazeDS, or ColdFusion Flash Remoting as a communications option. In ColdFusion 9, BlazeDS is installed automatically with the ColdFusion server. See Adobe's ColdFusion support web site at *www.adobe.com/support/coldfusion* for more information. Flash Builder 4.5 unlike its previous versions enables you to switch to a different application server for an existing project. Choose Project > Properties > Flex Server, select an application server, and configure it for use with the current project.

Henceforth we will assume that you've set the application server type to None until and unless specified. The next screen of the Flex Project wizard asks you to provide the Compiled Flex application location, also known as the Output folder. The default is a sub folder of the project root named bin-debug. This folder contains a compiled version of the application,

which you'll use for debugging and testing. The production version of the application is created in a separate step after the project has been created.



The 'New MXML Application' Window

When you create a project without an associated application server, the output folder is created as a subfolder of the project root. For Web applications, you then test the application by loading it into the browser from the local file system. For projects that do use an application server, the output folder is created within the document root folder of the testing Web server, and the application is loaded from the server with an HTTP request.

Components and Controls - Understanding Flash Builder's user interface

Flash Builder 4.5 adds unique tools to Eclipse to facilitate Flex application development. These tools include Editors and Views. Flash Builder includes two editors you can use when creating your Flex applications.

MXML Editor

The MXML editor is used to work with MXML files, whether they represent application files or custom components. When you open an MXML file that's associated with a currently open project, the file is always opened in the MXML editor. This editor has two different modes: Source mode and

Design mode. Whether the file opens initially in Design or Source mode depends on what mode you've used most recently on other files.

ActionScript Editor

The ActionScript editor is designed for editing files that contain pure ActionScript code. This editor is useful whether you're a Flex developer or a Flash developer, because both products now can use the latest version of the ActionScript programming language. When you open an AS file, the file is opened in the ActionScript editor. Both the MXML and ActionScript editors to make coding faster and more productive include Language color-coding, Auto-import of external ActionScript classes, Auto-completion of MXML tags and attributes, Auto-completion of variable symbol names, Code hinting for function arguments and class members, Intelligent language search for symbols and their declarations and Easy linking to consumed components and classes

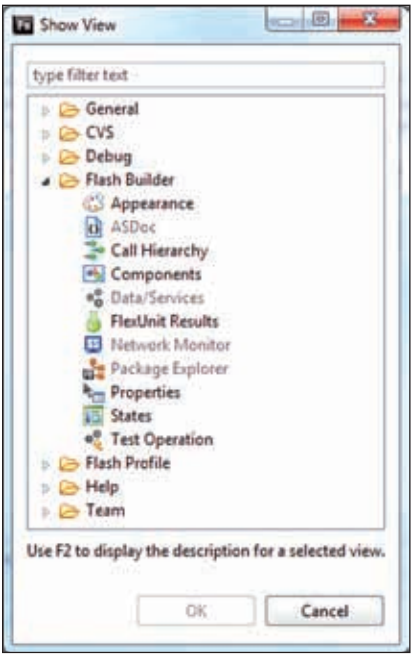
Flash Builder 4.5 has a Correct Indentation feature that enables you to automatically format your code. To use it, first select the section of ActionScript or MXML code you want to format, then select Source > Correct Indentation from the Flash Builder menu. For a shortcut, press [Ctrl] + [A] to select all of the current code file, and then press [Ctrl] + [I] to automatically indent the selected code. You can configure how indentation is managed in Flash Builder's workspace preferences. Select Window > Preferences from the Flash Builder menu, and then select Flash Builder > Indentation in the category list. Preferences are available in the primary Indentation screen, and also in its nested ActionScript and MXML screens.

Flash Builder views

Flash Builder 4.5 includes custom views that serve particular purposes.

Package Explorer view

The Package Explorer view, replaced the Flex Navigator view that was used in Flex Builder 3. The new view displays a tree of folders, files, and code packages and enables you to locate and open any project resource, but adds a listing of properties and methods you declare in your MXML and ActionScript code. This view is displayed by default in both the Flash and the Flash Debug perspectives. When using any of Flash Builder's perspectives, you can open the view by choosing Window > Package Explorer from the Eclipse menu. Notice that the Package Explorer view displays not just the appli-



The View Menu can be accessed from Window > Other Views ...

cation and class files, but also drills down to display a component's methods and properties. You can double-click any property or method to jump to that part of a component or class's source code.

You can create new project resources directly within the Package Explorer view by right-clicking on any project folder or package. From the context menu that appears, select the type of resource you want to create. The Package Explorer view can display code packages using one of two presentation styles, The hierarchical presentation (the default) displaying packages in a tree control, in a visual style similar to that used to present a directory structure or the flat presentation which shows all code packages as siblings and does not assume a parent child relationship between packages.

Follow these steps to switch between package presentation styles:

1. Click the View Menu button on the Package Explorer toolbar. This button appears as a downward-pointing arrow.
2. Choose Package Presentation > Flat to switch to flat presentation, or choose Package Presentation > Hierarchical to switch back to the default presentation.

Outline view

The Outline view, displays a tree of the objects that have been declared in an MXML or ActionScript file. This view is displayed by default only in the

Flash perspective. Choose Window > Outline from the Eclipse menu to open this view in any other perspective. The Outline view enables you to easily locate code representing any declared variable or object, whether the object has been declared in MXML or ActionScript. To locate code representing any variable or object using the Outline view, click the object in the view. The cursor in the current editor then jumps to that part of the code and selects the code that declares the object. When an MXML editor has focus, the default Outline view shows only those objects that are declared in MXML code. To see properties and methods that are declared inside a <Script> section, click the Show class view button in the Outline view's toolbar.

Problems view

The Problems view, displays current compilation errors and warnings. When your code contains a bug, the Problems view shows you the description of the problem (an error message), the resource containing the problem (a source-code file) the path of the resource (the folder containing the problem file) the location of the problem (the line number) and the type of problem. Keep only those projects open that you're currently working on. If you have the Build Automatically feature turned on (the default setting), Flash Builder recompiles all open projects whenever any source file in any of the projects has been modified and saved. If you have any remaining errors or warnings in projects you have open but aren't using, it slows Flash Builder's compilation process and keeps those errors and warnings in the Problems view until you fix them or close the project.

Design views

These views are used only when an MXML editor is in Design view. The Properties view enables you to set object properties through a simple user interface and generates the appropriate MXML code to represent your selections. The Appearance view allows you to edit some key styles of your MXML objects. The Components view enables you to drag and drop common user interface components, including Containers and Controls, into your application. The States view enables you to manage alternate view states through Design view and generates code to represent the new states.

Debugging views

These views are primarily used when debugging a Flex application. The Console view displays tracing information and other detailed debugging

messages. The Debug view contains controls for stepping through code, terminating a debugging session, and resuming a debugging session. The Variables view displays the values of all pre-declared variables that are currently in scope while application execution is stopped on a breakpoint. The Breakpoints view enables you to manage your breakpoints. The Expressions view enables you to evaluate and inspect arbitrary ActionScript expressions while application execution is stopped on a breakpoint. The Network Monitor allows you to monitor and introspect remote service calls. The new Network Monitor view enables you to monitor HTTP traffic sent between the Flex application and an application server at runtime.

Generating Code

Flash Builder 4.5 includes many features that enable you to generate and modify your application code. Some of these features have been part of the product since version 2, while others are new to Flash Builder 4.5

Generating getter and setter methods

ActionScript 3 has always supported the use of getter and setter methods to provide access to a class's private properties, but previous versions of Flash Builder didn't do anything to help developers build the required code. Getter and setter method generation is a reltively new feature added to the Flash Builder. You can now declare a public property and then use Flash Builder to convert it to a private property, supported by the required methods.

1. Create a new ActionScript class.
2. Declare a public property named myVar data typed as a String. The ActionScript class might start looking like this:

```
package
{
public class MyClass
{
public var myVar:String;
}
}
```

3. Place the cursor anywhere in the variable name.
4. Choose Source > Generate Getter/Setter. A dialog box appears that prompts you for options.
5. Accept all the dialog box's default settings and click OK.
   The public property is renamed with an underscore prefix and changed

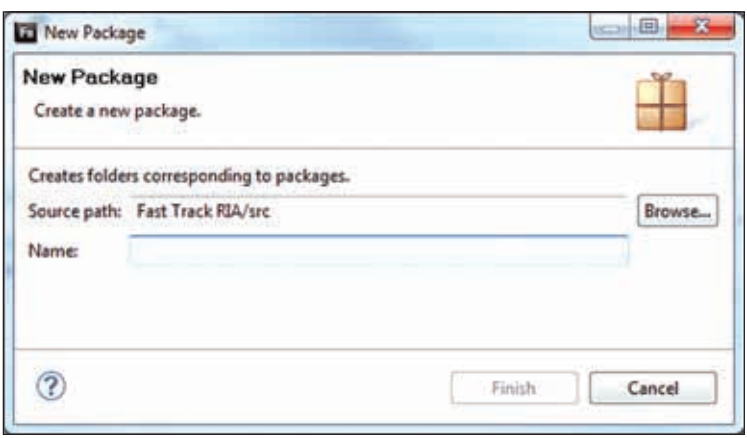
to a private property, and the required getter and setter methods are generated. The resulting class now looks like this:

```
package
{
public class MyClass
{
private var _myVar:String;
public function get myVar():String
{
return _myVar;
}
public function set myVar(v:String):void
{
_myVar = v;
}
}
}
```

You don't have to use getters and setters everywhere in your code (public properties do just fine for most purposes), but when you need them, Flash Builder now makes it much easier to build the basic code structure.

Skinning and styling

As described earlier, one of the Flex 4.5 SDK's most important new features



The New Package Window

is the capability to redefine the appearance of visual controls with programmatic skins. In past versions of the Flex SDK, you could create your own skin as an ActionScript class. By overriding certain methods that are called at runtime from the Flex framework and using the Flash drawing application programming interface (API) to declare vector graphics, you could define an entire visual presentation without ever using graphical applications like Adobe Photoshop and Illustrator.

This strategy, however, was slow and cumbersome. And, because the graphical applications mentioned earlier couldn't interpret the ActionScript code and show you a preview of its results, you had to use your imagination (and a lot of graph paper) to figure out how to code the image you wanted to create. In Flex 4.5, skins are now defined as MXML components that are extended from new classes named Skin and SparkSkin. You can bind the skin component to an application or custom component at compile time or runtime with its skinClass style. The skin class can be constructed with a combination of vector graphics, bitmap graphics, and Flex components to achieve whatever design ideas your application requires. We will see how to define custom skins using the new MXML architecture, and then how to attach them to your applications and custom components.

Creating and Using Spark Custom Skins

A custom skin component for a Flex 4.5 application or any of its Spark components is based on the new Skin and SparkSkin classes. You can start either by creating a new copy of a component's default skin class (a step made very easy by Flash Builder 4.5) or you can create a new fresh component that extends the Skin class. While it's possible to create such components in either MXML or ActionScript, most developers find that the MXML approach is much easier and the resulting code far more readable.

You can create a new custom skin using the new architecture for any component that includes the new SkinnableComponent class in its inheritance hierarchy. SkinnableComponent extends UIComponent, which means that Spark components can be added as display children of any Spark or MX containers. In this section, we will look at how to create a new custom skin for the Spark Application component, but most of the techniques also apply to all other Spark components.

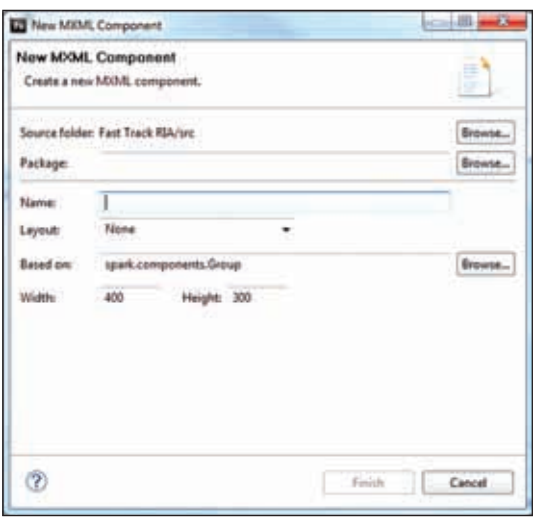
To create a new custom skin, start by creating a new custom MXML component. As with all MXML components and ActionScript classes, it is recommended that you place your custom skins in a specially named

package under your Flex project's source-code root folder. If you're using the Flex project that's available for download from the Web, you'll see that it already has a package named skins under the source-code root folder. If you're working on your own Flex project, you'll need to create this new package:

1. Open the Flex project in Flash Builder.
2. Right-click the default package in the Package Explorer view and choose New >Package.
3. Name the new package skins.
4. Click Finish.

Next create a new skin component:

1. In the Package Explorer, right-click the new skins package and choose New > MXML Component.
2. In the New MXML Component dialog box, type CustomAppSkin in the Filename field.
3. Accept the Layout default value of None.
4. Click the Browse button next to Based on.
5. Select spark.skins.SparkSkin and click OK.
6. Delete the default values in the Width and Height settings.
7. Click Finish to create the new skin component.



New MXML Component Window. Right click in the Package Explorer window to access.

The code for the new skin component looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<s:SparkSkin xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Declarations>
<!-- Place non-visual elements
(e.g., services, value objects) here -->
</fx:Declarations>
</s:SparkSkin>
```

The new skin component's code looks like that of any other custom component and can contain instances of any Flex component. Its difference lies in its superclass. Because its root element is <s:SparkSkin>, its purpose is to implement the component's visual design.

After creating the new skin component, you should associate it with the Spark component it's designed to modify. You create the association with the [HostComponent] metadata tag. This tag takes as its only attribute a string that describes the fully qualified package and name of the component:

```
[HostComponent("spark.components.Button")]
```

As with the [Event] metadata tag, [HostComponent] is placed inside the <fx:Metadata> element. Follow these steps to add the component association to the new custom skin:

1. Add a blank line under the <s:SparkSkin> starting tag.
2. Add the following code to the component:

```
<fx:Metadata>
[HostComponent("spark.components.Application")]
<fx:Metadata>
```

In your skin component, you'll be able to refer to the host component's properties using ActionScript code and binding expressions. You refer to the host object with an id of host Component, and to its properties using dot notation. For example, in a custom skin for the Application component, you could output the application's current height and width to the screen with the following Label and binding expression:

```
<s:Label text="{hostComponent.width},{hostComponent.height}"/>
```

In the next step, you declare view states to match the host component's required skin states. You don't have to refer to these view states in the component's visual design. For example, the Spark Application component has two required states. If you want the application to look the same in all view

states, you just set the skin's objects, properties, and styles without any state notation. But you must declare the states or a runtime error is generated when the skin is bound to the application at runtime.

First look at the host component's documentation to find out which states are required. Then open the Flex help system and search for spark.components.Application. After opening the help screen for the Application component, click Skin States. If necessary, click Show Inherited Skin States to see the states inherited from the Application component's superclasses. The Application component requires the skin states disabled and normal. Now return to the new skin component file in Flash Builder and add the following code after the default <fx:Declarations> element:

```
<s:states>
<s:State name="normal" />
<s:State name="disabled" />
</s:states>
```

In the skin code, you'll be able to designate inclusion or exclusion of particular objects depending on which state the application is in with the includeIn and excludeFrom properties. For example, the following Group is only included in the normal state:

```
<s:Group id="normalGroup" minWidth="0" minHeight="0"
includeIn="normal" >
... visual controls and graphics ...
</s:Group>
```

You're also able to control individual properties and styles with dot notation. The following style overrides the skin's default chromeColor setting only when the application's state is disabled:

```
chromeColor.disabled="#EEEEEE"
```

Spark components typically implement one or more required skin parts. These are typically used for specific functionality in the component. For example, the Application component requires a skin part named contentGroup that must implement the Group component or one of its subclasses. This is the actual container used to lay out the visual children declared with MXML in the Application. Or in button skins, the labelDisplay skin part is a TextBase object that will display the label of any button that uses that skin. If you don't declare the required skin parts in your custom skin, the application fails at runtime when it tries to use the skin part; in the case of contentGroup, it fails when it tries to add visual controls declared with MXML to the group upon application startup.

As with skin states, you can look at the documentation to find out which skin parts are required by the host component. To declare the Application component's required contentGroup skin part in CustomAppSkin.mxml, place the cursor above the closing </s:Application> tag and add the following code:

```
<s:Group id="contentGroup"/>
```

At this point, the custom skin should look like,

```
<?xml version="1.0" encoding="utf-8"?>
<s:SparkSkin xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Metadata>
 [HostComponent("spark.components.Application")]
</fx:Metadata>
<fx:Declarations>
<!-- Place non-visual elements
(e.g., services, value objects) here -->
</fx:Declarations>
<s:states>
<s:State name="normal" />
<s:State name="disabled" />
</s:states>
<s:Group id="contentGroup"/>
</s:SparkSkin>
```

The custom skin doesn't implement a visual design yet, but it has everything you need to make it compatible with the Spark Application component. The next step is to add visual objects to the skin. Because the custom skin uses the default basic layout, objects can be overlaid on top of each other, and you can use constraint-based layout properties such as top, left, bottom, and right to position objects relative to the skin's borders.

Visual objects that are added to the layout in the host component should appear in the application's background. To make this happen, just declare any FXG graphics in the skin component's code above the declaration of the contentGroup object.

To add a background color and a frame with a drop shadow to the application's appearance, in CustomAppSkin.mxml, place the cursor above the Group with an id of contentGroup and make a new blank line. Then declare a rectangle that covers the entire application canvas and has a solid light gray color of #CCCCCC:

```
<s:Rect width="100%" height="100%">
<s:fill>
<s:SolidColor color="#cccccc"/>
</s:fill>
</s:Rect>
```

Add another rectangle after the first, but still before the contentGroup object. This rectangle is constrained to 20 pixels from all application borders, uses rounded corners, and has a drop-shadow filter:

```
<s:Rect left="20" right="20" top="20" bottom="20"
radiusX="15" radiusY="15">
<s:fill>
<s:SolidColor color="#eeeeee"/>
</s:fill>
<s:filters>
<s:DropShadowFilter/>
</s:filters>
</s:Rect>
```

The custom skin now creates a framed appearance. Next you'll add settings to ensure that visual objects that are added to the application from the host component are placed within the visual area created by the frame. You do this by adding constraint properties to the contentGroup object in the custom skin. Now, add left, right, top, and bottom properties to the contentGroup object, setting all values to 30. This causes any visual objects added to the Application to be placed within the rounded rectangle with extra padding of 10 pixels on all borders:

```
<s:Group id="contentGroup"
left="30" right="30" top="30" bottom="30"/>
```

Binding a custom skin to a Spark component

You bind a custom skin to a Spark component using the component's skinClass style. This style is declared in the SkinnableComponent class and is therefore available to all of its subclasses. The skinClass style's type is set to Class, but the class must extend UIComponent. Skins designed for use with Spark components should extend SparkSkin. As with all styles, you can set the skinClass style in many ways. You can set it as an inline declaration for a particular component instance or in a style sheet to affect all instances of a particular component or at runtime using the setStyle() method.

You assign a skin to a single component instance at compile time by setting skinClass with an inline style declaration. When assigning a skin to the Application component, this strategy makes the most sense, since by definition there is only one instance of that component. The value of the skinClass style should be set to the fully qualified package and name of the custom skin component. To use the custom skin that was created in the previous section, set the <s:Application> tag's skinClass attribute to skins.CustomAppSkin.

You can also apply custom skins with style sheet declarations. While this strategy is most useful when applying a skin to a group of components (such as all objects that are instances of a particular Spark component), it can be used with any Spark component, including Application. As with all styles, the declaration starts with a CSS selector that defines which components will have the style applied. You can use type selectors, style name selectors (also known as class selectors), contextual selectors, or any other syntax that's supported in the Flex 4.5 SDK. The value of the skinClass style is set using the ClassReference() compiler directive. You pass the fully qualified package and name of the custom skin component. The following code assigns a skin named MyCustomButtonSkin to all instances of the Spark Button component:

```
s|Button: ClassReference("skins.MyCustomButtonSkin");
```

You can also load a custom skin at runtime using the setStyle() method or any of the techniques available with the StyleManager class. For example, the following code adds the custom skin to the application:

```
this.setStyle("skinClass", skins.CustomAppSkinComplete);
```

You can also get a reference to the class being currently used as a skin using the getStyle() method:

```
var currentSkin:Class = this.getStyle("skinClass") as
Class;
```

Skinning other Spark components

You can create custom skins for all Spark components, including your own custom components. In this section we describe the specific code you need to skin the Spark Button and List components. In Flash Builder, you can get started with custom skinning by creating a copy of a component's default skin class. This tool is available only from Flash Builder's design mode.

1. Open AppWithButtons.mxml in Flash Builder. Notice that the application uses a completed version of the custom application skin created in a

previous section of this chapter and declares a number of Button objects laid out vertically and centered:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
skinClass="skins.CustomAppSkinComplete">
<s:layout>
<s:VerticalLayout paddingTop="20"
horizontalAlign="center"/>
</s:layout>
<s:Button label="Button 1"/>
<s:Button label="Button 2"/>
<s:Button label="Button 3"/>
<s:Button label="Button 4"/>
</s:Application>
```

2. View the application in Design mode.
3. Right-click the first button and select Create Skin
4. Set Package to skins.
5. Set Name to CustomButtonSkin.
6. Check the option to Remove ActionScript styling code.
7. Click Finish. The new custom skin should be created and opened in Design mode.
8. Click the Source button to look at the new skin component's code.

Assigning custom skins with CSS

As described in a previous section of this chapter, the skinClass attribute is an inline style declaration that styles only the current Button instance. If you want to style all instances of the Button class throughout the application, first remove the skinClass attribute from the MXML code you used to create the custom skin, and then declare skinClass in a style sheet that affects the entire application. Open AppWithButtons.mxml in Source mode and remove the skinClass attribute from the first Button object. Then add the following code above the first <s:Button> declaration:

```
<fx:Style>
@namespace s "library://ns.adobe.com/flex/spark";
s|Button {
skinClass: ClassReference("skins.CustomButtonSkin");
}
</fx:Style>
```

Views and effects

Flex applications define view states as particular presentations of a visual component. In each moment of the user's interactions with the application, each visual component presents itself in a particular form known as its current view state. Flex enables you to define as many different view states as you like for the Application and for each of its custom MXML components using declarative code, and then enables you to switch easily between states by setting the Application or custom component's currentState property. View state management in Flex is designed primarily for application scenarios where the Application or component uses a significant portion of its presentation in multiple situations and makes incremental changes to its presentation for each new situation. This sort of incremental change is different from application navigation, where the user moves between multiple different layers, or views, that don't share content with each other. In Flex 4.5, you always declare view states in MXML. In Flex 3, you could also use ActionScript code to declare view states, but the resulting code was verbose and difficult to maintain. Even further, the MXML view state code has been dramatically altered since Flex 3, resulting in much cleaner MXML declarations.

When you switch view states at runtime, you can make the change abruptly or, using transitions, you can choreograph the change with ActionScriptbased effects. A transition is a class that enables you to easily associate effects with view state changes. View states are used to define incremental changes to an existing view. For example, a login form that initially requests a user name and password can, with the addition of a few more controls, also be used as a registration form. Any visual objects you declare are by default included in the form's initial presentation. To create different versions of the form, you declare new view states with unique names.

View states are declared with MXML code. You can either code a view state manually or, using Flash Builder's Design mode, generate the required code based on changes you make to a component at design time. View states are identified by creating new State objects and setting their name property, which is a String value that you assign in MXML code. Each visual component that's displayed on the screen in a Flex application has an initial view state that's named by default State1. The default state is defined by all the object's current property and style settings, event handlers, and in the case of containers, nested child components in its display list, and it's represented by the main MXML in the document.

In Flex 4.5, the default value of the current State property is the name of

the first State object declared in the states property (an array). Flex state names cannot include periods, spaces, or other special characters. This ensures that the code used in MXML declarations to set attributes for particular states is syntactically correct. For example, this Label is only bold in a state named "boldState":

- `<s:Label text="Hello World" fontWeight.boldState="bold"/>`

If the state name were allowed to be set as "bold.State" (with the period), the code to set the property would be:

- `<s:Label text="Hello World" fontWeight.bold.State="bold"/>`

The parser would fail on this declaration.

Defining View States in Design View

Flash Builder has a States view that shows up by default only when the Flex Development perspective is active and the current application or component is being edited in Design view , the States view has a toolbar that includes the buttons New State, Duplicate State, Edit State and Delete State

You can create a new view state by clicking the New State button on the toolbar, or by right-clicking in the States view and selecting New State from the context menu. The New State dialog box, asks for the properties,

- **Name -** A non-blank String value is required.
- **Duplicate of -** This asks which state the new view state is based on.
- **Blank State -** This selection results in the removal of any existing visual objects from the new state. You can easily change this in the code after the state has been created.
- **Set as start state -** This check box enables you to assign the new state as the application or component's starting state upon instantiation.

Defining a view state's overrides

To define a view state's override actions in Design view, first select the appropriate view state from the States view or the State selector. Then make changes to the Application or component with these design time actions:

- Add components to the state by dragging from the Components view.
- Remove components by selecting and deleting them.
- Change components' properties and styles through the Flex Properties view.
- Change event handlers through the Flex Properties view.

You can select the current view state either from the States view or, from the view state selection menu in the Design view editor toolbar. When working with an application or component whose width or height exceeds the available dimen-

sions of the Design view editor, you may not see scrollbars appear. Remember that you can use the Zoom and Pan tools to move around the design surface.

Follow these steps to make incremental changes to the application:

1. Reopen ViewStatesBegin.mxml in Flash Builder's Design mode.
2. Using either the States view or the State selector, set the current view state to oneway.
3. Select the SimpleText control with the text value of Return, and press Delete to remove the control from the view state.
4. Select the DateField control in the same area of the application, and delete it as well.
5. To test the view states in Design view, set the current state to State1. You should see that the two controls you deleted are displayed in their original locations.
6. Switch back again to the oneway state. You should see that the two controls are removed from the current display.

Switching view states at runtime

All visual components support the currentState property to enable switching from one view state to another at runtime. The value of currentState is a simple String, so this code switches the current Application or component's state to oneway:

```
this.currentState = "oneway";
```

To return to the default state, set currentState to its name:

```
this.currentState = "State1";
```

Try the following steps with the airfare search application described in the preceding section:

1. Reopen ViewStatesBegin.mxml in Flash Builder, and switch to Source view.
2. Locate these RadioButton control declarations:

```
<mx:RadioButton label="Round Trip" selected="true"/>
<mx:RadioButton label="One Way"/>
```

3. Add a click event handler to the RadioButton control labeled One Way that changes the currentState to oneway:

```
<s:RadioButton label="One Way"
click="currentState='oneway'"/>
```

4. Add a click event handler to the RadioButton control labeled Round Trip that changes the currentState to State1 (the default state):

```
<s:RadioButton label="Round Trip" selected="true"
click="currentState='State1'"/>
```

5. Run the application, and click the RadioButton controls to switch between view states. You should see that the two controls appear and disappear as you click the RadioButton controls to switch view states.

Declaring view states in MXML

A view state is represented by the <s:State> tag set and is always assigned a name:

```
<s:State name="myNewState">
... state declaration ...
</s:State>
```

To declare one or more view states in a main application or custom component file, wrap the <s:State> declaration tags within <s:states> tags:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<s:states>
<s:State name="State1"/>
<s:State name="oneway"/>
</s:states>
... declare visual objects ...
</s:Application>
```

The <s:states> element must always be declared as a child element of the Application or component root; you cannot nest state declarations within other child MXML tag sets unless they are in an <fx:Component> tag set (used with custom item renderers.

Technically speaking, it doesn't matter whether the <s:states> tag set is at the top, bottom, or middle of the MXML code, as long as it's a direct child of the MXML file's root element and doesn't separate visual components from each other. Once a state has been declared, you can then indicate which controls are a part of the state with the includeIn or excludeFrom attributes. You can also modify properties, styles, and event handlers by adding new MXML attributes to the affected component declarations.

Managing view states in components

You can declare a view state inside a custom component. The rules are the same as for a main application file: You can only apply the view state to the entire component, not to its nested child objects. You can then control that

component's currentState either internally or from the component instance parent object. Remember that code within a custom component uses this to refer to the current instance of the component. The following code switches the currentState of the component instance to a new state:

```
this.currentState = 'myNewState';
```

Using transitions

Transitions are a way of associating animations, implemented as Flex effects, with runtime changes from one view state to another. By default, when you switch to a view state that changes the visibility, size, or position of objects on the screen, the change is visually abrupt. A transition enables you to slow down and choreograph the change so that it's easier and more fun to watch. As with view states, you typically declare transitions using MXML code. Each visual component has a transitions property. The transitions property is an Array containing one or more instances of the Transition class. To declare transitions in MXML, you create an <s:transitions> tag set as a direct child of the main application or component file's root element, often right after the <s:states> declaration. Then nest as many <s:Transition> tag sets within <s:transitions> as you need:

```
<s:Application...>
<s:states>
...declare <s:State> elements here...
</s:states>
<s:transitions>
...declare <s:Transition> elements here...
</s:transitions>
... declare visual objects here ...
</s:Application>
```

In the Flex 4.5 SDK, the State and Transition classes are still members of the MX package mx.states, so they're sometimes declared with an mx prefix. However, the Flex SDK declares these classes in both the MX and Spark namespace manifests, so you can use either namespace prefix. Each transition is declared as an <s:Transition> tag with the properties 'fromState', the name of the starting state and 'toState' the name of the ending state. Each of these properties can be set to either an explicit state name or a wildcard (*), the default for both properties, to indicate that the transition applies to all state changes. You then indicate which animation you want to play by nesting the appropriate effect class within the <s:Transition> tag set. The effect should have its target or targets property set to indicate which objects should be animated. This transition is applied by

moving from a state named state1 to a state named state2. It has the effect of applying a Fade effect to an object that's being added in the state:

```
<s:transitions>
<s:Transition fromState="state1" toState="state2">
<s:Fade target="{addedObject}"/>
</s:Transition>
</s:transitions>
```

You also can use Parallel or Sequence effects to introduce more complex animation. The Parallel effect combines multiple effects that execute simultaneously. The following transition causes a Move effect and a Zoom effect to play simultaneously, creating the visual effect of an object "exploding" from the top-left corner of the application into its final position:

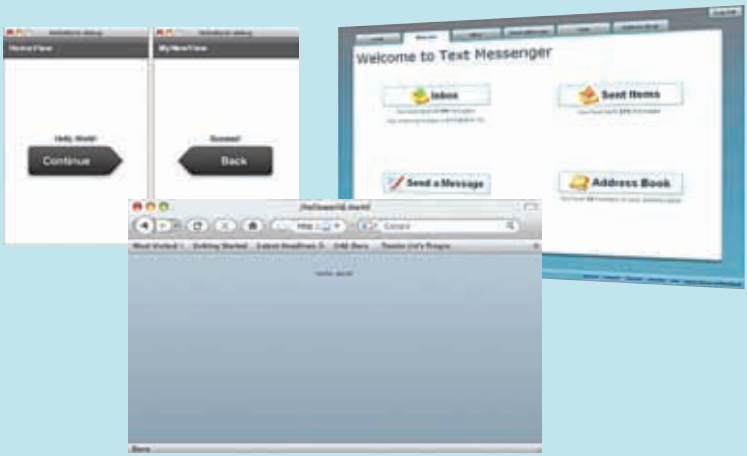
```
<s:Transition fromState="*" toState="detail">
<s:Parallel target="{detailImage}">
<s:Move xFrom="0" yFrom="0" xTo="300" yTo="50"/>
<s:Zoom zoomHeightFrom="0" zoomWidthFrom="0"
zoomHeightTo="1" zoomWidthTo="1"/>
</s:Parallel>
</s:Transition>
```

As described earlier in this chapter, the Sequence effect enables you to play multiple effects consecutively. The following code declares a Transition with a Sequence that executes a horizontal Move effect first, and follows it with a vertical Move:

```
<s:Transition fromState="*" toState="detail">
<s:Sequence target="{detailImage}">
<s:Move xFrom="0" yFrom="0" xTo="300" yTo="0"/>
<s:Move xFrom="300" yFrom="0" xTo="300" yTo="300"/>
</s:Sequence>
</s:Transition>
```

The effects framework includes the following special classes that are designed to control when an object is added or removed from a presentation or when other actions are taken relative to when particular visual effects are executed in a transition,

- **AddAction** - Defines when an object is added to the presentation.
- **CallAction** - Calls a function of the defined target object.
- **RemoveAction** - Defines when an object is removed from the presentation.
- **SetAction** - Sets the value of a named property of an object.



YOUR FIRST FLEX APP

All you'll need, to create an app for the web using Adobe Flex

In this chapter, we look at how to create and deploy a basic "Hello World" Flex application for the Web. After the application is built, we will see the fundamental nature of a Flex application, including the relationship between the application SWF file and the supporting HTML (Hypertext Markup Language) files. We will be describing the contents of the HTML "wrapper" file that's generated for you in Flash Builder and its associated JavaScript library file.

Finally, we will see how to deploy the Flex application into a Web site as a distinct application that opens in its own window, as an applet that's displayed as part of an existing Web page and as a desktop application deployed on Adobe AIR. By the end of this chapter, you should have a good sense of what a Flex application is and how it's delivered to the user.

Note that the code samples and screen shots in this chapter assume that you're using Flash Builder 4.5 to build the application. If you're using the Flex 4.5 SDK and your own text editor, the steps will be similar, but you won't have access to some of the code completion and other productivity tools described.

Hello World

In all programming languages, your first task is to write a "Hello World" application. The simplest of applications, it typically contains no more than a single line of text output.

Throughout these instructions, we will assume that you're using the stand-alone version of Flash Builder. Where the steps are different in the plug-in version, we provide alternative steps. Your first step is to create a Flex project. The project hosts the application's source code and other assets.

Follow these steps to switch to a new workspace:

1. Open Flash Builder 4.5
2. From the menu, choose File and then click Switch Workspace.
3. Select a new folder named flexprogram anywhere on your hard disk and then click OK. If you're working on Windows, the folder might be C:\flexprogram. On Mac OS X, the folder should be in your home directory; for example, /Users/[username]/Documents/flexprogram.

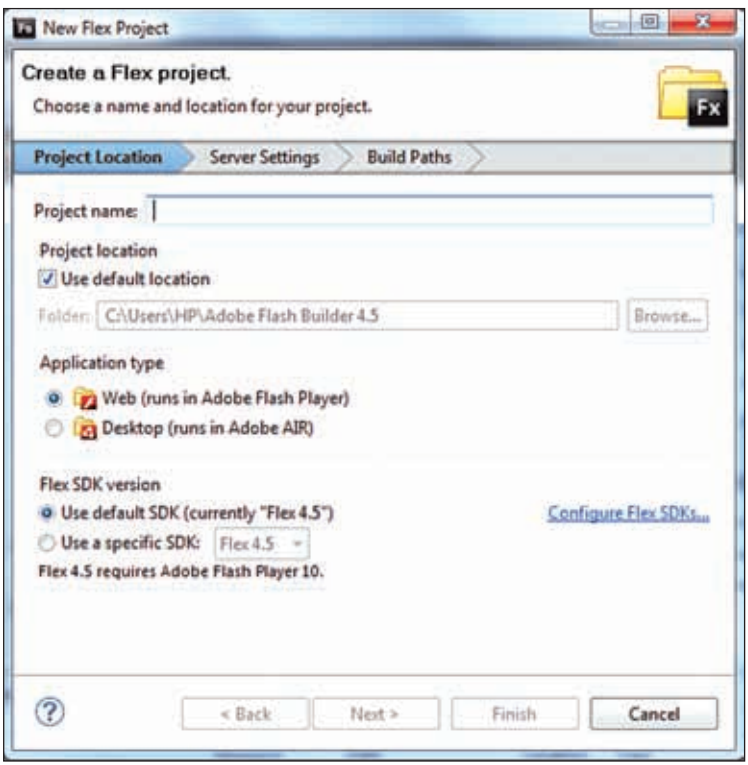
After selecting the workspace, you should see Flash Builder close and reopen. The new workspace should display the Flash Builder Start Page and the default Flash perspective. The newly created workspace is empty and contains no projects. Workspace folders are sometimes created as sibling folders to the projects they reference, rather than parent folders. This is because a workspace contains references to an absolute location on the hard disk and isn't portable. If you change the location of your project folders, you have to re-create the workspace.

You can re-create a workspace from scratch by first closing Flash Builder and then deleting the workspace folder's .metadata subfolder using Windows Explorer or Finder. When you reopen Flash Builder, the workspace data is recreated automatically. You'll then need to import any existing projects to see them in the Package Explorer view. Choose File > Import to open the Import dialog box. Then choose General > Existing Projects into Workspace. Browse and select the project's root folder. After verifying that your project is visible and has been selected, click Finish.

Creating the project

Follow these steps to create a Flex project:

1. From the menu, choose File > New > Flex Project. If you're using the plug-in version of Flash Builder, choose File > New > Other. Then from the wizard that appears, choose Flash Builder > Flex Project.
2. In the first screen, enter hello as the Project name.
3. Confirm the Use default location option under Project location is selected. The project location defaults to a folder named chapter03 under the workspace folder. For example, on Windows the default folder might be C:\flexprogram\hello.
4. Choose the Web application (runs in Flash Player) option for the Application type.
5. Choose the Use default SDK (currently "Flex 4.5") option for the Flex SDK version.

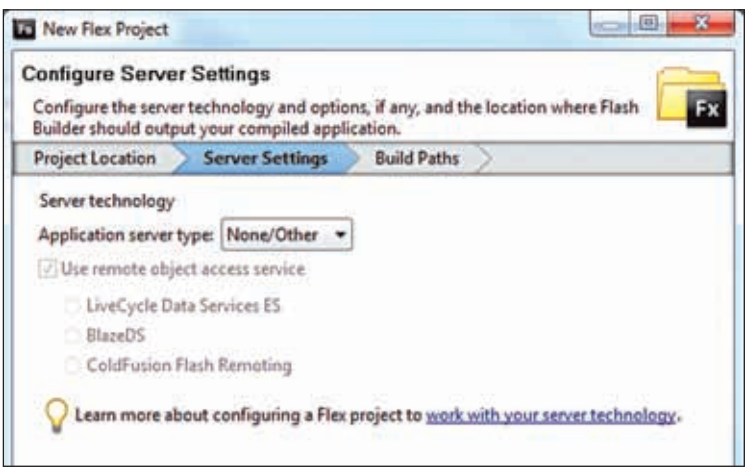


'New Flex Project' Window

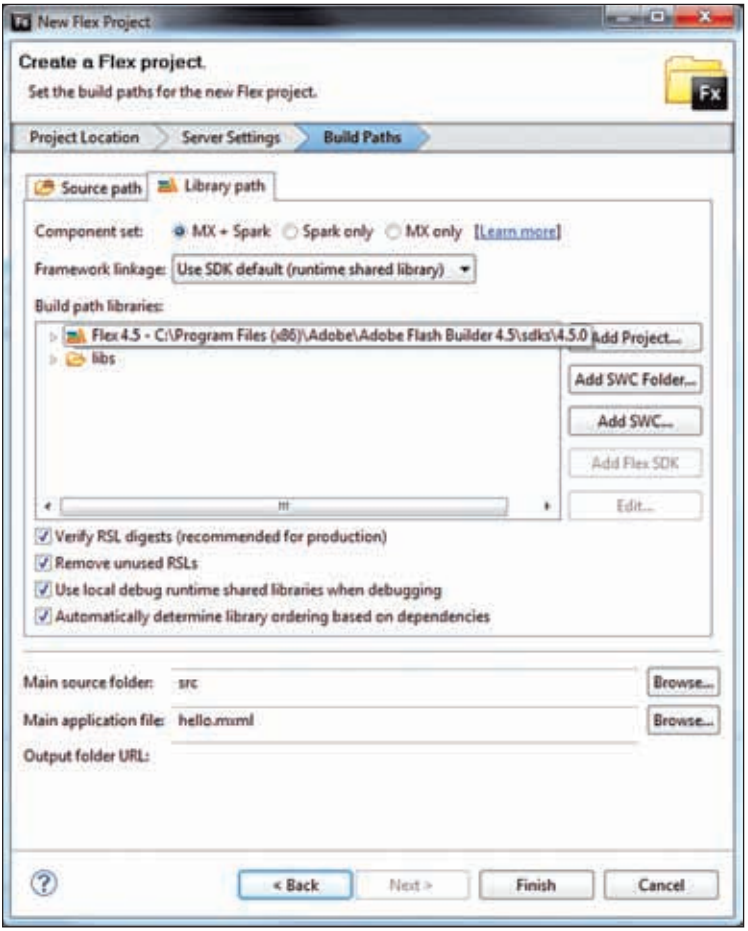
6. Select None/Other from the Application server type drop-down menu, and click Next.
7. On the Configure Output screen, accept the Output folder setting of bin-debug. This is the location of the compiled debug version of the application and its supporting files.
8. Click Next.
9. On the Create a Flex project screen, accept these default settings:
- Main source folder: src.
- Main application file: HelloWorld.mxml.
- Output folder URL: Accept the default setting, leaving it blank.
10. Click Finish to create the project and the main application file. You should see the main application file appear in the Editor view. If you're working in a completely new workspace, the file should appear in Source mode; that is, you should see the application's source code.

Saying hello

Flex 4.5 applications use a new architecture for laying out the application's child objects. In Flex 3, the <mx:Application> component had a layout property that was set to a String value of absolute, horizontal, or vertical. In Flex 4.5, the <s:Application> tag's layout is determined by an instance of a Layout class. You typically set the application's layout property with an



Second Screen of the New Flex Project window



Third Sceen of the New Flex Project window

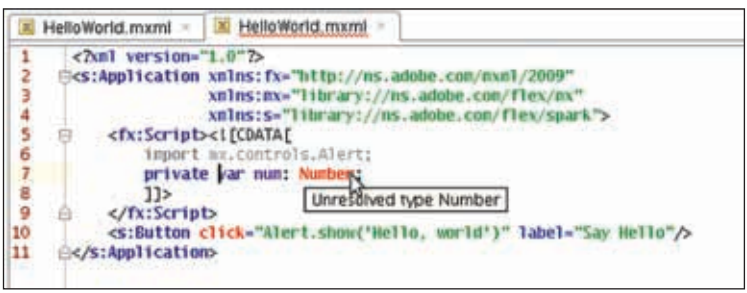
<s:layout> tag, wrapped around an instance of the layout class you want to use. The following code sets an application to use vertical layout:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
minWidth="955" minHeight="600">
<s:layout>
```

- `<s:VerticalLayout/>`
- `</s:layout>`
- `</s:Application>`

The default layout for a Flex 4.5 application is a scheme known as basic layout. This architecture is similar to Flex 3's absolute layout, which caused visual objects to retain their positions relative to the application's top-left corner. In your simple application, you'll use vertical layout.A simple Flex 4.5 application uses more XML namespaces and MXML child tags. Follow these steps to display a simple message in your Flex application:

1. Make sure your application is displayed in Source mode.
2. Delete the minWidth and minHeight attributes from the <s:Application> tag. This enables the application to automatically resize itself to adjust to the browser's dimensions.
3. Delete the <fx:Declarations> element and its nested comment. This element is used to declare nonvisual objects in Flex 4.5 but isn't required in this simple application.



The Code Hinting available in Flash Builder 4.5

4. Place the cursor between the <s:Application> tags.
5. Type <layout. As you type, you should see that a window that contains a list of proposed properties and objects you can use in this context.
6. Select s:layout from the list, and then type > to close the tag. You should see that Flash Builder auto-completes the <s:layout> tag:

- `<s:layout>`
- `</s:layout>`

7. With the cursor between the <s:layout> tags, type <vertical. Select s:VerticalLayout from the list, then complete the tag with XML empty tag syntax:

```
<s:layout>
<s:VerticalLayout/>
</s:layout>
```

8. Add a paddingTop setting of 20 and a horizontalAlign setting of center to the <s:Vertical> tag:

```
<s:VerticalLayout paddingTop="20"
horizontalAlign="center"/>
```

9. Place the cursor on an empty line of code between the </:layout> end tag and the </s:Application> end tag.
10. Type <label, then select s:Label from the class list.
11. Complete the Label object as follows:

```
<s:Label text="Hello World" fontSize="36"/>
```

12. Check your completed application. The code should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<s:layout>
<s:VerticalLayout paddingTop="20"
horizontalAlign="center"/>
</s:layout>
<s:Label text="Hello World" fontSize="36"/>
</s:Application>
```

13. Choose File > Save to save the file.
14. Choose Run > Run HelloWorld to run the application in a browser. As shown in, you see that the application opens in a browser window.

Understanding the html-template Folder

Each Flex project that's designed for deployment over the Web contains a folder named html-template. This folder contains models for the HTML and supporting files that run your application in the browser. Whenever you save changes to your source code, Flash Builder automatically rebuilds your application using the HTML model file to generate an HTML wrapper. At the same time, it copies the contents of the html-template folder to the output folder that contains the compiled application.

During the compilation process, most of the files in the html-template directory are simply copied to the output folder that contains the debug version of the project's applications. The HTML wrapper file that you use

at runtime is generated based on a model file in html-template named index.template.html. In addition, any files in the source-code root that aren't either code files built in MXML or ActionScript, or that don't represent embedded assets, such as binary graphical files, are copied, along with their directory structure, to the output folder as well.

The html-template directory contains three files and one subfolder:

- **index.template.html -** A model file that is the basis for the generated HTML "wrapper" files that call the application at runtime.
- **swfobject.js -** A JavaScript library containing functions used at runtime to load Flash Player. This file also contains "sniffer" code that can discover whether Flash Player is currently loaded on the user's desktop and, if so, which version.
- **playerProductInstall.swf -** A Flash application that's used to upgrade a user's system when Flash Player 6.65 or higher is installed.
- **The history subfolder.** Contains files to implement the history management feature (for non-IE browsers only):
  - historyFrame.html is a model for an HTML page that's loaded into an <iframe> in the main page at runtime.
  - history.js is a JavaScript library containing functions that are called by historyFrame.html.
  - history.css contains Cascading Style Sheet (CSS) rules to suppress the visibility of the history frame in the main page.

The HTML wrapper template

The HTML template file, index.template.html, contains a combination of HTML code, calls to JavaScript functions that are stored in swfobject.js, "History" files that manage the navigation history & deep linking and placeholders for values that are passed to the generated version of the file. At compile time, the HTML template is used to generate an HTML "wrapper" file that you deploy to your Web server.

The HTML <head> section

The <head> section of the model HTML file contains links to a set of CSS and JavaScript files. The <title> element contains a variable that's filled in from the Application's pageTitle property:

```
<title>${title}</title>
```

To fill in this value in the generated HTML wrapper page, set the pageTitle property in the <s:Application> start tag:

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
pageTitle="Hello World">
```

Next, a CSS declaration eliminates default margins and paddings, and sets the height of the <html> and <body> elements to 100%. The flashContent id selector refers to a element that determines where the application is displayed in the Web page:

```
<style type="text/css" media="screen">
html, body { height:100%; }
body { margin:0; padding:0; overflow:hidden; text-
align:center; }
#flashContent { display:none; }
</style>
```

The overflow setting of hidden means that if the size of Flash Player (or another element in the page) overflows the boundaries of the page, the remainder is hidden. If you want the page to show scrollbars instead, change the value of the overflow style to scroll.

A <link> tag incorporates the history.css file from the history folder, and a <script> tag imports the history.js JavaScript library:

```
<link rel="stylesheet" type="text/css" href="history/his-
tory.css" />
<script type="text/javascript" src="history/history.js"></
script>
```

This <script> element imports the swfobject code library:

```
<script type="text/javascript" src="swfobject.js"></script>
```

Then, another <script> element contains code that evaluates the user's current version of Flash Player and reacts accordingly:

- If the version of Flash Player that's required by your application is present, the application is loaded.
- If the user has at least Flash Player 6,0,65, but not your required version, the Flash application playerProductInstall.swf is loaded to assist the user in upgrading his player software. If any errors are encountered (if the user doesn't have administrative rights to his computer, for example), the Flash-based upgrade installer fails with a useful error message (rather than just hanging and letting the user wonder what happened).
- If Flash Player isn't available, or a version older than 6,0,65 is present, a link to download Flash Player from the Adobe Web site is presented.

This section of the HTML document's JavaScript code defines which version of Flash Player is required for the current application:

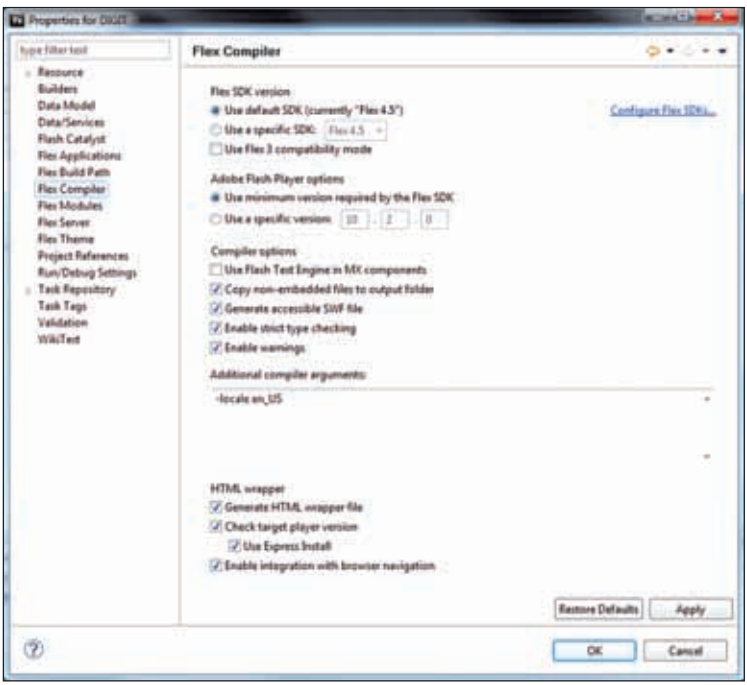
- `var swfVersionStr =`
- `"${version _ major}.${version _ minor}.${version _ revision}";`

The version_major, version_minor, and version_revision parameters can be set in the Flex project's properties:

1. Choose Project > Properties from the Flash Builder menu.
2. In the Properties dialog box, select the Flex Compiler section
3. In the Required Flash Player version option, change the version numbers as needed.

When you create a new Flex Project in Flash Builder 4, the project's required Flash Player version is determined by the Flex SDK's configuration. This setting is in the SDK folder's frameworks/flex-config.xml file, in the <target-player> element:

- `<target-player>10.0.0</target-player>`



Setting the required Flash Player version through the Flash compiler option under Project > Properties

If you change the contents of flex-config.xml, you should "clean" all projects to force recompilation. Choose Project > Clean, select Clean all projects, and click OK. The rest of the script completes the version evaluation and runs the application:

```
var xiSwfUrlStr = "playerProductInstall.swf";
var flashvars = {};
var params = {};
params.quality = "high";
params.bgcolor = "${bgcolor}";
params.allowscriptaccess = "sameDomain";
var attributes = {};
attributes.id = "${application}";
attributes.name = "${application}";
attributes.align = "middle";
swfobject.embedSWF(
"${swf}.swf", "flashContent",
"${width}", "${height}",
swfVersionStr, xiSwfUrlStr,
flashvars, params, attributes);
<!-- JavaScript enabled so display the flashContent div in
case it is
not replaced with a swf object. -->
swfobject.createCSS("#flashContent",
"display:block;textalign:
left;");
```

The call to swfobject.embedSWF() dynamically generates HTML code that runs Flash Player and the application at the location of the with an id of flashContent. The call to swfobject.createCSS() hides the remainder of the tag's contents so doesn't display if the application runs successfully. The <head> section contains the tag with an id of flashContent:

```
<div id="flashContent">
<p>
To view this page ensure that Adobe Flash Player version
${version _ major}.${version _ minor}.${version _ revision} or
greater is installed.
</p>
<a href="http://www.adobe.com/go/getflashplayer">
<img src="http://www.adobe.com/images/shared/download _
buttons/get _ flash _ player.gif" alt="Get Adobe Flash Player"
```

```
/>
</a>
</div>
```

Its contents are only displayed if the application or the Flash upgrade installer aren't loaded successfully. By default, it displays a text message indicating which version of Flash Player is required and a linked image that takes the user to the Adobe Web site to download the latest version of Flash Player. You can customize this section of the HTML code as desired.

Deploying the Application

You've created the application, and it runs beautifully in your development and testing environment. Now you want to share the application with your users. This section describes how to create a version of the application that's suitable for public release and make the application available to your audience.

Creating a release build

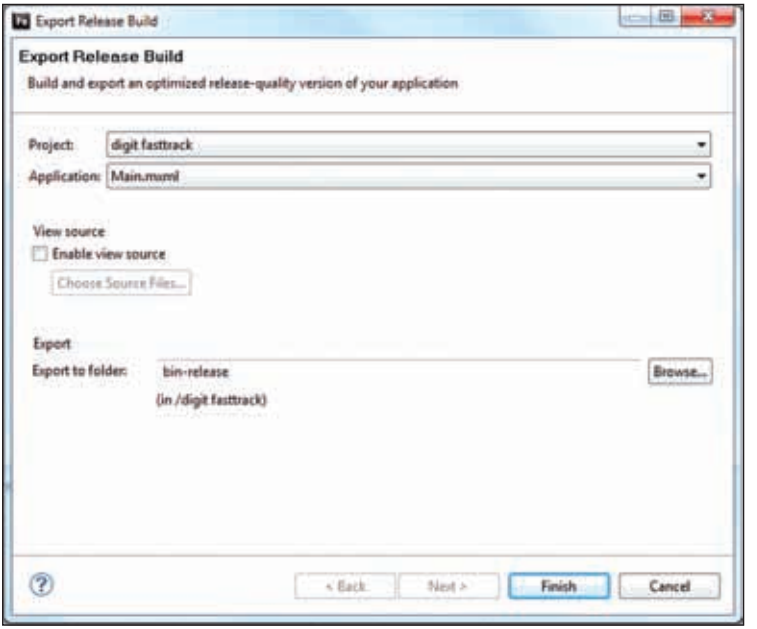
The version of the application that's created in your output folder, and that you run during the testing and debugging phase of development, is the "debug" version of the application. This compiled SWF file is significantly larger than the version you'll ultimately deploy for your users, because it contains additional internal information that's used only during debugging.

In Flex Builder 2, the debug and release builds of the application were placed in a single output folder. To deploy the application, you copied all files except the HTML and SWF files with the word debug in their filenames to the Web server. Starting with version 3, Flex Builder separates the debug and release builds into separate folders and requires a manual export process for the release build.

To create a release build of a Flex Web application, follow these steps:

1. From the Flash Builder menu, choose Project > Export Release Build, or File > Export > Release Build.
2. In the Export Release Build dialog box, make these choices:
   a. Select the application you want to export.
   b. Indicate whether you want to enable the View Source feature.
   c. Select a folder to which you want to export the release build.
3. Click Finish to export the release build.

A release build folder contains only a single application and its supporting files. In contrast, the bin-debug folder contains the debug versions of all applications in a project. After exporting the release build, you should



The Export Release Build dialogue box for a web application

have a new folder containing the compiled application and its supporting files. This version of the application is optimized for delivery to the user. It doesn't contain debug information, and as a result it's significantly smaller than the debug version.

The size of a basic "Hello World" compiled application file with a single Label control will be either 98K for the debug version, or 51K for the release build. Clearly, you want your users to be downloading and using the release build.

In Flex 4.5, the ActionScript classes and other elements of the SDK that are shared by all Flex applications are compiled by default into RSL (Runtime Shared Library) files that are separate from the application SWF file. These files, which also have a file extension of .swf, are loaded by the application at runtime as their classes are needed. Examples of these files include framework_4.0.0.12685.swf, spark_4.0.0.12685.swf, and so on. (The specific version number embedded in the filenames changes depending on which version of the Flex SDK the application is compiled with.) When you deploy a Flex 4 .5 web application, you must copy all of the SWF files in the release folder to the application folder on your Web site.

You can change this behavior in the Framework linkage drop-down menu on the Flex Build Path screen of the project options dialog box. To cause the SDK classes to be compiled into the main application SWF files, set Framework linkage to Merge into code. After saving your changes, select Project > Clean from the Flash Builder menu, then click OK to rebuild the project's applications. The resulting application SWF file will be much larger, but you'll have fewer files to upload to the web site.

Testing the release build

You can test the release build of a Flex Web application by opening its HTML wrapper file in a Web browser. Here's how:

1. From the Package Explorer view, open the release build folder and locate the HTML wrapper file. This file has the same name as the application itself, but has a .html file extension.
2. Right-click the HTML file, and choose Open With > Web Browser.

The application opens in a Web browser nested with an Eclipse editor view. When you run the release build as described previously, the application always opens from the local file system, rather than from any Web server you might have configured. If you need to test the application with a Web server, you have to manually configure the server, or place your bin-release folder within your Web server's document root folder, then open the file from a Web browser using the appropriate URL.

Deploying the release build

To deploy the release build of the application, just upload all files in the release build folder to your Web site using File Transfer Protocol (*FTP)* or whichever method you typically use to deploy other files to your Web site. These files will include the following:

- The compiled application SWF file
- The SWF files containing framework classes, used by the application as RSL's
- The HTML wrapper file
- The JavaScript library file
- playerProductInstall.swf
- The history folder
- Any assets added to your application
- Then provide the URL of the HTML wrapper page to your users.

Programmers sometimes make users navigate to a Flex application in a

new browser window. The new window then has a fresh "history," which means the browser's Back button is disabled and the user can't accidentally unload the application by trying to go back to a previous screen. The following HTML code would open the application from the home page of a web site

```
<a href="registration/registration.html" target="_blank"/>
```

Integrating an application into an existing Web page

Some Flex applications are designed to be presented as applets (an application that represents only part of a Web page). This is easy to accomplish if you have some working knowledge of HTML.

1. Create a region of a Web page where you want to host the application. Design it just as you would to host an image, an ActiveX control, or a Java applet. You can use HTML tables or more modern tags with CSS to control the size and position of the hosting region.
2. In the Flex application code, set the <s:Application> tag's height and width to a specific number of pixels that will make the application size match the available space in the Web page. For example, if you have a tag in the hosting page that's 300 pixels high and 200 pixels wide, use this code in the Flex application to size it appropriately:

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
height=»300» width=»200»>
```

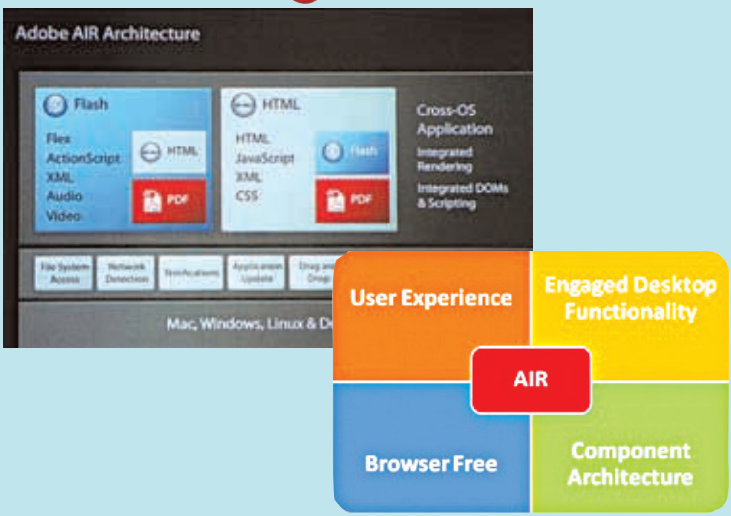
When the application is compiled, the height and width settings are passed into the generated HTML file.

3. Copy the JavaScript includes and initialization code from the <head> section of the generated HTML wrapper file to the <head> section of the hosting HTML page.
4. Create a element with an id of flashContent in your Web page where you want to display the application.

When you deploy a hosted Flex applet to a Web server, be sure to include all the same files as before: the JavaScript library, history files, and upgrade installer SWF file (playerProductInstall.swf).

CHAPTER #6

ADOBE® AIR™



DESKTOP AIR APPLICATION USING FLEX

Create your own desktop apps using Adobe Flex

thinkdigit.com

Adobe's release of Flex 3 in February 2008 was tightly integrated with the release of Adobe AIR. Formerly known as the Adobe Integrated Runtime (and before that by its public code name, Apollo), AIR is Adobe's strategy for offering a universal runtime client that can run local applications on a variety of personal computer systems and other computing devices. With AIR 1.0, Adobe delivered the capability to deploy applications on Windows, Mac OS X, and Linux client systems. With the release of Flash CS5 Professional, Adobe is extending support for AIR to all major smart phones. The roadmap for AIR includes future versions for other cell phones and mobile devices, which eventually would allow AIR desktop applications to be deployed on a more truly universal basis.

AIR applications can be built from many different kinds of assets, but each application's core asset is made up of either Flash-based content, built in either Flash Professional (starting with Flash CS3) or Flex, or HTML-based content. Regardless of which kind of asset is used as the application's core element, any AIR application can use and present HTML, Flash, Flex, or Acrobat PDF content.

The Adobe AIR 2.0 SDK, which includes many new features for integration with local clients, can be downloaded from the Adobe Web site at: *www.adobe.com/go/air.* In this chapter, we describe the basics of creating and deploying a Flex based desktop application with Flash Builder 4.5 and Adobe AIR.

AIR Architecture

Adobe AIR is installed as a runtime library on your client system. Its purpose is to provide core runtime functionality that's needed by all AIR-based desktop applications, regardless of whether they're built in Flash, Flex, or HTML. Adobe AIR includes a copy of both the Flash Player and a Web browser. AIR 2.0 includes Flash Player 10.1, while the Web browser is an implementation of WebKit 4.0, an open-source Web browser engine. A desktop application deployed on AIR is delivered as an installable archive file with a file extension of .air. After installation, it runs as a local application that's native to the operating system, rather than as a Web-based application. As a result, desktop applications deployed on AIR aren't subject to the same security sandbox restrictions as a Web-based application that's downloaded and run on request from within a Web browser.

Because an AIR application's assets are made up of content that runs equally well on multiple operating systems without having to be rebuilt (Flash documents, HTML pages, JavaScript, CSS code, and Acrobat PDF documents), a single application can run on all supported operating systems without having to be recompiled.

If you're using Flash Builder 4.5 to develop Flex applications, you don't necessarily have to install AIR on your development system because Flash Builder includes all the tools you need to compile, test, and debug an AIR application. But to fully install a completed application, the runtime must be installed.

You can install the runtime in two ways,

1. If you know you need AIR on your system, you can download the AIR installer from Adobe's Web site and install it on your system prior to installing any applications.
2. When you install an AIR application that uses a seamless installation badge, the application installer detects whether the runtime is already installed and, if not, offers to include the runtime installation along with the application.

Creating a Flex desktop application

You can create and deploy a desktop application with Flex using one of these strategies:

1. If you're using the free Flex SDK to build your Flex applications, you can use the free AIR SDK to package your applications for deployment.
2. If you're using Flash Builder to create your Flex applications, everything you need to package an AIR application is already included.

In this section, we will look at the steps for building a Flex desktop application project with Flash Builder. When you create a new Flex project in Flash Builder, you have the option of setting the Application type to Desktop. All MXML applications in such a project are designated as desktop applications and are tested and deployed with AIR.

In Flash Builder, you can't deploy a single application to both the desktop and the Web. The selection of Air-based or Web-based deployment is made at the project level, and after a project is configured as such, you can't change it without going back and rebuilding the project from scratch. If you do need to create a Flex application that's deployed with both architectures, consider creating three projects: one for the Web, one for the desktop, and one that's created as a Flex library project. The first two projects would have

applications that are bare skeletons and get all their real functionality from components in the library project. Then, as you code and compile the library project, its assets are shared with the "real" projects that contain and are responsible for building the Web and desktop applications.

Follow these steps to create a Flex project that's designed for the desktop in Flash Builder:

1. Choose File > New > Flex Project from the Flash Builder menu.
2. On the first screen of the New Flex Project wizard, set the project properties:

- Project name: Fast Track
- Project location: Use default location
- Application type: Desktop
- Flex SDK version: Use default SDK
- Application server type: None

3. Click Next.
4. On the Configure Output screen, accept the default Output folder location and click Next.
5. On the final screen, set the Main application filename to MyDesktopApp.
6. Set the Application ID to com.mycompany.MyDesktopApp.
7. Click Finish to create the project and application.
   You should see a starting Flex application open in the Flash Builder editor with a root element of <s:WindowedApplication>. In the Package Explorer view, you should see both the main application file and the XML-based application descriptor file MyDesktopApp-app.xml.
8. Delete the default <fx:Declarations> element from the new application.
9. Add a Label control to the application with a text property of Hello from AIR! and a fontSize style of 24. Place it 20 pixels from the top of the application and center it horizontally:
   <s:Label text="Hello from AIR!" fontSize="24"
   top="20" horizontalCenter="0"/>
10. Save and run the application.

The appearance of a basic Flex application differs depending on the hosting operating system. The magic of AIR is that you don't have to recompile the application for different operating systems. When packaged with default settings in the application descriptor file, the version of AIR that's currently hosting the application at runtime controls the application window's look and feel.

Packaging a release version of an AIR application

When you package a release version of an AIR application, you create an AIR file (with a file extension of .air) that's delivered to the user as the application installer. When the user opens the AIR file on a system where Adobe AIR has already been installed, the application installer is executed.

Follow these steps to package the application for installation and deployment:

1. With the application open in the Flash Builder editor, choose Project > Export Release Build from the menu.
2. In the Export Release Build wizard, set these properties:

- Project: The selected project.
- Application: The MXML application you want to package.
- View source: Whether you want to allow the user to view the application's source code (available when the user right-clicks on the application at runtime).
- Export to file: The name of the generated AIR file you want to build. By default, this file is placed in the Flex project's root folder, but you can browse and select any other location within a currently open Eclipse project.

3. Click Next.

On the Digital Signature screen, you can either export and sign the generated AIR file with a digital certificate or create an intermediate file with a file extension of .air that can be signed and completed in a secondary step.

To package any AIR application, you must provide a security certificate that certifies to the user who developed the application. For applications that are in testing or that are only deployed within an organization, you can generate a self-signed certificate from within Flash Builder. This certificate enables you to package and deploy the application, but because no recognized certificate authority will have authenticated your organization's identity, the resulting installer application indicates that the author of the application is "Unknown."

Even an application that reports an unknown author has unrestricted access to the user's system. The purpose of the security certificate is to give the user an opportunity to accept or reject installation based on the author's identity and doesn't stop bad applications from doing bad things. If you don't have a security certificate, follow these steps to create a self-signed certificate for testing or internal use:

1. Click Create on the Digital Signature screen.

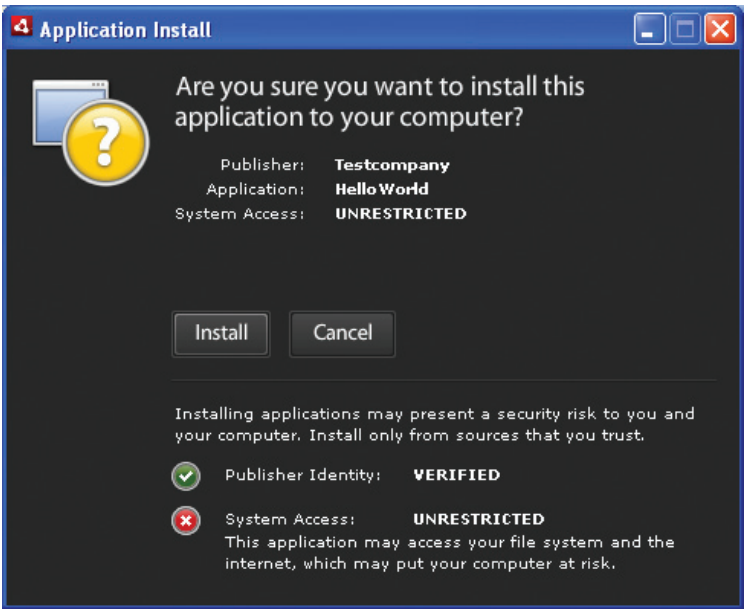
2. Enter the requested values on the Create Self-Signed Digital Certificate screen. Items marked with an asterisk are required. In particular, you must provide a password that will then be required each time the certificate is used.
3. Select the name of your certificate file with a file extension of .p12, and click OK to create the certificate file. When you return to the Digital Signature screen, the certificate filename and password will are already filled in. If you already have a digital certificate file, just select the file and enter the certificate password.
4. Click Finish in the Digital Signature screen to create the AIR installer file. You should see that the application's AIR file is available in the project root folder and can be seen in the Package Explorer view,

Installing AIR applications

To install an AIR application on a desktop system that already has the runtime installed, just open the AIR file that was generated in Flash Builder. From within Flash Builder, you can open the file by double-clicking it in the Project

The AIR Install Screen

Explorer view. The initial installation screen displays the application's Publisher (displayed as UNKNOWN when the AIR file is built with a self-signed certificate) and the application's name as configured in the descriptor file.



Installing a Hello World AIR app on a desktop

After clicking Install on the initial screen, the confirmation screen, the application name and the description as provided in the descriptor file. The installer also offers the user these options:

- Whether to include a shortcut icon for the application on the desktop
- Whether to start the application after installation is complete
- The application installation location, which defaults to C:\Program Files on Windows and /Applications on Mac OS X

On Windows, the application is installed in a subfolder of the selected location named for the application name. For example, the default location MyDesktopApp on Windows is a folder named C:\ProgramFiles\MyDesktopApp. On Mac OS X, the application is installed as a single application package file in the selected location folder with a file extension of .app. For example, the default location of MyDeskTopApp on Mac is a single application file named /Applications/MyDesktopApp.app.

Flex Application Tips and Tricks with AIR

As described earlier in this chapter, the subject of developing Flex applications for desktop deployment with AIR is too large for a single chapter. There are, however, a few specific things you do a bit differently in a desktop application, and there are many Flex SDK features that are available only when you're developing for AIR including Debugging AIR applications in Flash Builder, Rendering and managing HTML-based and PDF-based content, Using the WindowedApplication component as the application's root element and Creating channels at runtime for communicating with Remoting gateways

Debugging AIR applications in Flash Builder

For the most part, debugging an AIR application in Flash Builder is just like debugging a Webbased Flex application. You have access to all the same debugging tools, including the trace() function, breakpoints, and the capability to inspect the values of application variables when the application is suspended.

When you run a Flex application from within Flash Builder in either standard or debug mode, Flash Builder uses ADL (AIR Debug Launcher) in the background. In some cases, ADL can stay in system memory with hidden windows even after an AIR application session has apparently been closed. The symptom for this condition is that when you try to run or debug that or another application, Flash Builder simply does nothing. Because a debugging session is still in memory, Flash Builder can't start a new one.

Open the Windows Task Manager and in the Processes pane, locate and select the entry for adl.exe. Click End Process to force ADL to shut down and then close Task Manager, and return to Flash Builder. You should now be able to start your next AIR application session successfully. One common scenario that can result in this problem is when a runtime error occurs during execution of startup code. For example, if you make a call to a server-based resource from an application-level creation Complete event handler and an unhandled fault occurs, the application window might never become visible. If you're running the application in debug mode, you can commonly clear the ADL from memory by terminating the debugging session from within Flash Builder. When running in standard mode, however, the ADL can be left in memory with the window not yet visible. To solve this issue, it's a good idea to explicitly set the application's initial windows as visible. In the application descriptor file, the <initialWindow>

element's child <visible> property is commented out by default. Because this value defaults to false, if the window construction code never succeeds to a runtime error, you're left with an invisible window and ADL still in memory. To solve this, open the application's descriptor file, uncomment the <visible> element, and set its value to true:

```
<visible>true</visible>
```

Working with HTML-based content

The Flex framework offers two ways of creating a Web browser object within any application, One way is by the HTMLLoader class which is extended from the Sprite class and can be used in any Flash or Flex application. Because this class doesn't extend from UIComponent, you can't add it to a Flex container with simple MXML code or by using the addChild()or addElement() methods. The second way is through HTML control bring extended from UIComponent and can be instantiated with either MXML or ActionScript code.

The HTML control is quite a bit easier to use and provides the same functionality as HTMLLoader. Declaring an instance of the control results in a Web browser instance that can freely navigate to any location on the Web (assuming the client system is currently connected).

As with all visual controls, the HTML control can be instantiated in MXML or ActionScript code. After it's been instantiated, its location property determines which Web page is displayed. This HTML object, for example, displays Adobe's home page and expands to fill all available space within the application:

```
<mx:HTML id="myHTML" width="100%" height="100%"
location="http://www.adobe.com"/>
```

When you assign the HTML control's id property, you can then reset its location as needed from any ActionScript code. In addition to the location property, the HTML control implements these methods that enable you to control navigation with ActionScript code:

- historyBack() - Navigates back one step in the control's history list.
- historyForward() - Navigates back one step in the control's history list.
- historyGo(steps:int) - Navigates the number of steps. The value of the steps argument can be positive to move forward or negative to move back.

Using the WindowedApplication component

Flex applications designed for desktop deployment typically use

<s:WindowedApplication> as the application root element. A beginning desktop application's code looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/
mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Declarations>
<!-- Place non-visual elements (e.g., services, value
objects) here -->
</fx:Declarations>
</s:WindowedApplication>
```

The WindowedApplication component is extended from Application and provides all the application-level functionality you expect from a typical Flex application. It also adds these capabilities that are unique to Flex desktop applications:

- Native menus can be displayed and integrated into the overall application look and feel.
- The application can be integrated with a dock or system tray icon to provide easy access to common application functions.
- The application can display operating system-specific "chrome" (the graphics in the application window's border, title bar, and control icons).
- A status bar can be displayed at the bottom of the application window for string-based status messages.

Here's an example. The WindowedApplication component can display a status bar at the bottom of the application window. This display is controlled by two of the WindowedApplication component's properties, showStatusBar:Boolean, which is the property when true (bydefault), the application window displays a status bar and the status:String value displayed in the status bar. The following modified custom updateHTML() function from the NewTitlesReader application updates the application's status bar with the title of the currently selected RSS item:

```
private function updateHTML():void
{
myHTML.location = feedSelector.selectedItem.link;
status = "Current Title: " + feedSelector.selectedItem.
title;
}
```

Creating Remoting channels at runtime

When a Web-based Flex application communicates with an application server that supports Flash Remoting (known as the Remoting Service in LiveCycle Data Services and BlazeDS), it typically uses a channel definition with dynamic expressions that evaluate at runtime to the location of the server from which the application was downloaded. This is the default my-amf channel delivered with BlazeDS:

```
<channel-definition id="my-amf"
class="mx.messaging.channels.AMFChannel">
<endpoint
url="http://{server.name}:{server.port}/{context.root}/
messagebroker/amf" class="flex.messaging.endpoints.AMFEndpoint"/>
</channel-definition>
```

The <endpoint> element's url attribute uses dynamic expressions to evaluate the server name and port and the context root of the hosting instance of BlazeDS. This approach doesn't work with desktop applications deployed with AIR, because the concept of "the current application server" doesn't have any meaning in a desktop application. Instead, you must provide the explicit location of the server-based application to which Remoting requests should be sent at runtime.

A conclusion on Adobe AIR

In addition to the features described in this chapter, AIR applications can accomplish the following tasks that aren't possible with Flex Web applications:

- Full screen and spanned monitor display
- Integration with native visual components such the operating system's window and menu systems
- Creation of applications with transparency that serve as widgets
- Reading and writing files and folders on the local file system
- Persisting data in SQLite, a local database embedded in the runtime
- Interacting with the system clipboard, including drag-and-drop to and from AIR applications and the OS
- Synchronization of data managed on the server by LiveCycle Data Services
- Access to all network services supported by the Flex framework

The subject of building and deploying AIR-based desktop applications is worthy of an entire Fast Track, and in fact there are many such books available.

APPLICATIONS FOR MOBILE DEVICES USING FLEX

The power to create mobile apps, across platforms!

The Adobe Flash platform, in general, now comprises a varied set of integrated technologies that empower and enable developers to create rich internet applications that can be run consistently and efficiently across multiple mobile devices working on different platforms. In this chapter, we will be discussing the various tools and workflows that can be used for the development of cross platform functional mobile applications within the Flash platform and the different devices and OS platforms you can aim at targeting. It is important to that this is only with respect to what we can do with Flash today. Soon we might be able to do much more. During the Mobile World Congress 2011 in Barcelona, Adobe

announced very impressive numbers about the growth and momentum of mobile devices market and a lot of it was accredited to the Adobe Flash Player and platforms providing support for Adobe AIR applications. To quote a few points, there were at least 20,00,000 Flash Player 10.1 installs in the first six months after its launch, statistically accounting for around 12 per cent of the smartphones shipped last year and the market survey by Strategy Analytics estimates that in 2011 alone more than 132 million smartphones will have Flash Player support, representing at least 36 per cent of all smartphones being sold worldwide.

By the end of last year, developers and content authors across the globe were deploying Adobe AIR applications in over eighty four million mobile devices running Android and iOS. This number is expected to rise dramatically to at least over two hundred million by the end of this year. During the initial two months after the release of AIR on the Android platform, more than 1,500 applications were created and were available for download on the Android Marketplace. Today, it is estimated that at least 3 million Flash Platform developers are employing Adobe Creative Suite 5 tools to create content and build applications across a substantial number of devices which use either the Flash Player or AIR. Adobe also in the meantime is working extra hard to improve its offerings and make cross platform mobile application development even easier, so you expect a lot more developments in the near future.

Because of the increasing demand for building applications for mobile devices a need for a simplified unified methodology has emerged. You'll see that Adobe AIR and its APIs are a big step in this direction. The major difference between AIR apps for mobile and the legacy AIR apps created for desktops is that for mobile app development you can use only ActionScript or Flex currently, unlike for desktop applications where you can make use of HTML, JavaScript, and CSS as well.

There are three main primary categories of applications that you can you build using AIR, ActionScript, and Flex for mobiles devices:

Games

The same old Flash games that we so love playing on the web and on our desktops can now be run inside an AIR app that can be packaged into mobile devices. For example, the Green Hornet Crime Fighter is an old Flash game that has also been converted into an AIR app for mobile. Kongregate also offers several Flash games which have optimised for AIR in a mobile. They even have support for GPU acceleration if required.

RIAs and enterprise apps

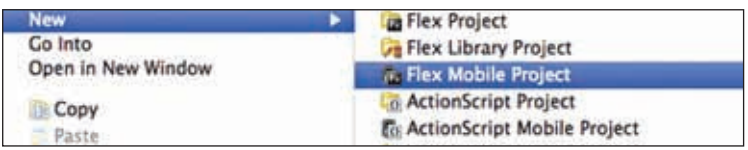
As discussed in the first chapter can also leverage existing services easily and then be turned into mobile applications. MAX Companion is an excellent RIA example. Similarly, so are the Acrobat Connect and the Standard Chartered Bank enterprise apps. We will now look at Flash Platform development for Android, BlackBerry, and iOS devices. Adobe's latest update to Flash Builder 4.5 has significantly boosted developer options and now supports publishing to Blackberry Tablet OS , iOS, and Android, thereby catering to the whole gamut of smartphone and tablet platforms currently available.

Android

Back in October 2010, Adobe first launched Adobe AIR 2.5 on the Android Market for all v2.2 "FroYo" devices. Adobe also launched a preview release of all the next generation tools that are in the market today which can be used for developing applications for mobile device using the Flash Platform: The newer versions Flash Builder 4.5, Flash Catalyst 5.5, and the Flex 4.5 framework is what we will be looking at for mobile application development.

Once you have Flash Builder 4.5 installed on your computer, you can create Android apps by using either Flex or ActionScript. You can make use of the Flash Builder to develop your application, test it on your computer using any of the emulators already provided by Adobe, deploy it on your Android device using a USB cable, and debug it if necessary while it's running on the device. After this entire process you can even submit the application to the Android Market. You can make use of the Flash Builder Export Release feature to build the APK file. On the other hand you can use Flash Professional CS5 for this as well.

Note that just like on a desktop, on an Android, the AIR applications use the AIR runtime which needs to be installed on the device. Imagine a scenario where an android user tries installing an app on his device without the runtime already installed on it. Just like on a desktop, it will not work without the runtime, but Flash prebuilds a feature into the package which



Starting a new Flex mobile project in the Flex 4.5 SDK

lets the installation process proceeds as usual but when the user tries to start the application, a notification will appear that required for you to install Adobe AIR to use the application and when user clicks on the install button that pops up, the runtime is installed like any other Android application. There is also a question of whether it were possible to simply package the AIR runtime along with your AIR application. Unfortunately though at this time, the answer is no. But you never know what features will come out in the future versions.

The Flex 4.5 release lets you easily build and create Android apps by putting in new components that were specifically designed to make the app work on mobile devices and by optimising existing components already present in previous versions. For example, Flex now provides support for screen management, like your application can have the UI split across multiple screens with only one visible and you can slide through or access them using buttons.

When building AIR for Android apps, API support is already available for the following areas

- Screen orientation / full-screen
- Session cache support restore app state
- Accelerometer
- StageWebView
- Multi touch and gestures
- Camera and microphone
- Hardware buttons
- Android permissions for Internet, SD card, GPS, camera, and so on

Tour de Mobile Flex and Adobe AIR Launchpad

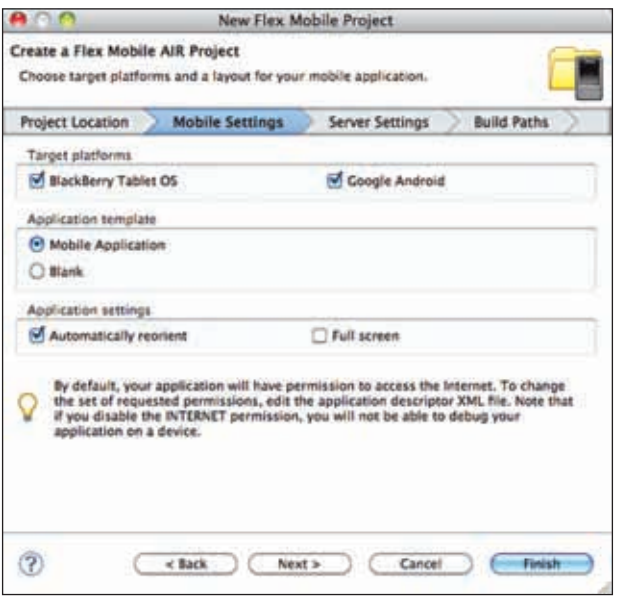
- Phone/SMS/e-mail/browser/Android Market integration
- GPS
- Local databases SQLite

And last but not the least you should be aware of two existing apps that can aid you in getting started with Android development. Tour de Mobile Flex is an Android app that has been built using AIR and the Flex 4.5 SDK. The second one is Adobe AIR Launchpad which is a desktop application that lets you create a Flex project which is ready for importing into the Flash Builder with all the required permissions for Android pre-set.

BlackBerry Tablet OS

With RIM's announcement of the new BlackBerry PlayBook, pitted against Apple's iPad, the Flash platform had to be extended yet again. Similar to what has been said about AIR for Android development, everything stands true for BlackBerry PlayBook development as well.

RIM took an immediate step to integrate AIR as a core part of the OS of this tablet device making Flash developers even happier. Despite this fact there are still a few minor differences between Android development using AIR and Blackberry table OS app development for AIR.



Creating a new Flex mobile AIR project for Blackberry Tablet OS

You will currently need to use the BlackBerry Tablet OS SDK and simulator to make these apps. After you've downloaded and installed them, you're ready to start developing apps with Flash Builder 4.5. The latest SDK has a BlackBerry Torch browser in it, so that the developers can not only test their AIR apps but also test their Flash web sites to see how it would look on the tablet browser. In case you choose to make your application look like the native BlackBerry PlayBook apps, a few of which have actually been created using AIR you can use the BlackBerry Tablet OS SDK for Adobe AIR. This SDK not only provides a library of UI classes that are same as the classes used for the PlayBook UI itself.

The SDK also lets you package and deploy PlayBook applications .bar files to the PlayBook simulator that can be run on your desktop. So basically making use of the SDK you can access the battery level or get a notification directly when a low memory event occurs. You can get more details on these APIs, from the 'BlackBerry Tablet OS SDK for Adobe AIR API Reference' available online. So to put it simply, you can develop applications using Flex 4.5 or just ActionScript for Blackberry devices as well. Just as in the case of Android app development, you can develop apps for Blackberry by using Flash Professional CS5 also. Adobe has been constantly working with RIM to make it easier and easier to build apps for the BlackBerry PlayBook using Flex.

iOS devices

If you look back at the history of what we call 'smartphones' in modern days and their application development with Adobe products, iOS was the first mobile platform that Adobe supported. Adobe had announced it in 2009 at Adobe MAX and subsequently released the Packager for iPhone in 2010 as part of the Adobe CS5 launch. If you want to develop applications for iPhone, iPod touch, and iPad, you can now use Adobe Flash Builder 4.5, and Flex 4.5, which, with their rich feature set let you deploy apps across the iPhone and iPad, without you having to bother about form factor or screen resolutions.

You can debug the application within Flash Builder, rather than the usual misery of installation and reinstallation of the app on the device. You can also debug directly on the device; however, the iPhone, iPod or iPad must be connected on the same Wi-Fi network as the computer on which the debugger is running. Similar to the Blackberry Tablet OS, with the June update of Flash Builder 4.5.1, you can also package applications built with Flex 4.5 and AIR code base.

A few areas where API support is available while targeting iOS devices is

- Cut, copy, and paste
- Accelerometer
- Multitouch
- Saving images to the photo library
- Screen orientation
- Geolocation
- Microphone access

You also cannot execute the code directly from a SWF loaded during the runtime as there is no virtual machine to do this; however you can make use of assets such as images or simple animations. So if you want to create an iOS app, the AIR runtime needs to be compiled together with your application code into the iOS native code for your app to work. This is the primary difference between how iOS apps and other apps developed for Android or BlackBerry work; this is because of the fact that for Android and Blackberry, the AIR runtime can be directly installed on the device itself, unlike iOS where Apple does not allow it.

ADOBE GETS SOCIAL!

Adobe India is here with its official Facebook page. Share all your technical encounters with us & learn about what's new and what's lined up.

To know more about us, share or exchange any cool information, simply log on to the Adobe India Facebook page. Be a part of the group & look out for all the latest updates at Adobe.

www.facebook.com/AdobeIndia

RNI REGN. No. MAH ENG/2001/3791

Flash Builder 4.5 & mobile applications

Adobe® Flash® Builder™ 4.5 Premium

Build for Android™, BlackBerry® Tablet OS and iOS mobile platforms using a common code base

New Adobe® Flash® Builder™ 4.5 software adds powerful mobile development capabilities, plus time-saving code editor improvements:

- Reach more devices with less code
- Accelerate Flex application development
- Work in parallel with designers

To order or upgrade to Adobe® Flash® Builder™ 4.5 Premium

Call: 1800 102 5567 | **Email:** iisadobe@adobe.com

ADOBE® FLASH® PLATFORM